September 1992

Solutions for High Density Applications Using Intel Flash Memory

MARKUS A. LEVY
DALE ELBERT
APPLICATIONS ENGINEERING
INTEL CORPORATION

Order Number: 292079-004

Solutions For High Density Applications Using Intel Flash Memory

CONTENTS

CONTENTS

INTRODUCTION

Mass storage encompasses many different technologies. Though commonalities exist, mass storage strives for nonvolatility, low cost per bit, and high density. Disk drives provide the best known example. However, many environments now require higher performance and reliability with lower power consumption, even at the expense of capacity. Flash memory uniquely meets these demands.

Flash memory can be used as a mass storage medium in applications including factory automation, notebook computers, high-end workstations, point of sale terminals, and data acquisition systems. Even desktop computers benefit from solid-state storage. The motivation to incorporate flash memory in any of these applications becomes obvious to the system designer who understands flash memory's benefits and density projections.

In an effort to understand these benefits, this document includes both conceptual and application oriented discussions. These discussions will be kept to a minimum with the real focus being on specific design techniques and considerations.

ADVANCED PACKAGING

Mass storage is synonymous with high density. Disk drives have increased the bit density of the rotating media via material improvements and closer tolerances. For semiconductors, density requires advanced packaging as well as higher capacity silicon (improved photolithography). Intel's Flash Memory devices are based on the company's EPROM Tunnel Oxide (ETOX™) technology that enables the high degree of scaling required to achieve high density.

Intel offers the high density flash memories in several package types. The standard packages are the Plastic Dual In-line Package (PDIP), the Plastic Leaded Chip Carrier (PLCC), and the Thin Small Outline Package (TSOP). Advanced modular packaging in the form of PCMCIA compatible memory cards and Single In-line Memory Modules (SIMM) provide the total solution.

Which package is best for your application?

Plastic Leaded Chip Carrier (PLCC)

The engineer striving to reduce board space is already using surface-mounted technology, such as PLCC. The PLCC is seen frequently on PC add-in cards and motherboards. Compared to the DIP, PLCC uses as little as 35% the overall board space. Its small size, compared to the DIP, is attributed to the terminal center-to-center spacing—50 mils versus 100 mils—as well as its four-sided pinout. No drilling or lead-cutting is necessary as leads are soldered directly to pads on the circuit board. The PLCC's 50-mil pad pitch is compatible with most circuit board manufacturing equipment. Additionally, components can be mounted on both sides of the board. However,the four-sided PLCC generally requires the use of a multi-layered board to lay out conductor traces for maximum compaction.

Thin Small Outline Package (TSOP)

When overall space constraints are critical, the TSOP is the best choice. This is best exemplified by IC memory cards. Low height is the key attribute of the TSOP, measuring 1.2 mm versus 3.5 mm for the PLCC. (Mechanical drawings in Appendix.) State-of-the-art center-to-center terminal spacing of 0.5 mm yields a smaller package and narrower conductor traces than the PLCC or DIP. In comparison, the volume of the TSOP is 172.8 mm^3 versus 656.3 mm^3 for the PLCC and 1872.3 mm^3 for the DIP.

The TSOP is available in standard and reverse pin configurations (Figure 1). Pins are located on only two ends of the package. This approach simplifies trace layout while reducing the number of board layers because traces can be routed out the non-leaded sides of the devices. Very dense board layouts are accommodated because components can literally be laid out end-to-end and side-by-side. Figure 2 displays an optimal layout best utilizing the TSOP's attributes. The close spacing allows one bypass capacitor to be used for two devices (provided they are not simultaneously selected). This optimal component layout can be mirror-imaged through the board to easily double the memory capacity.

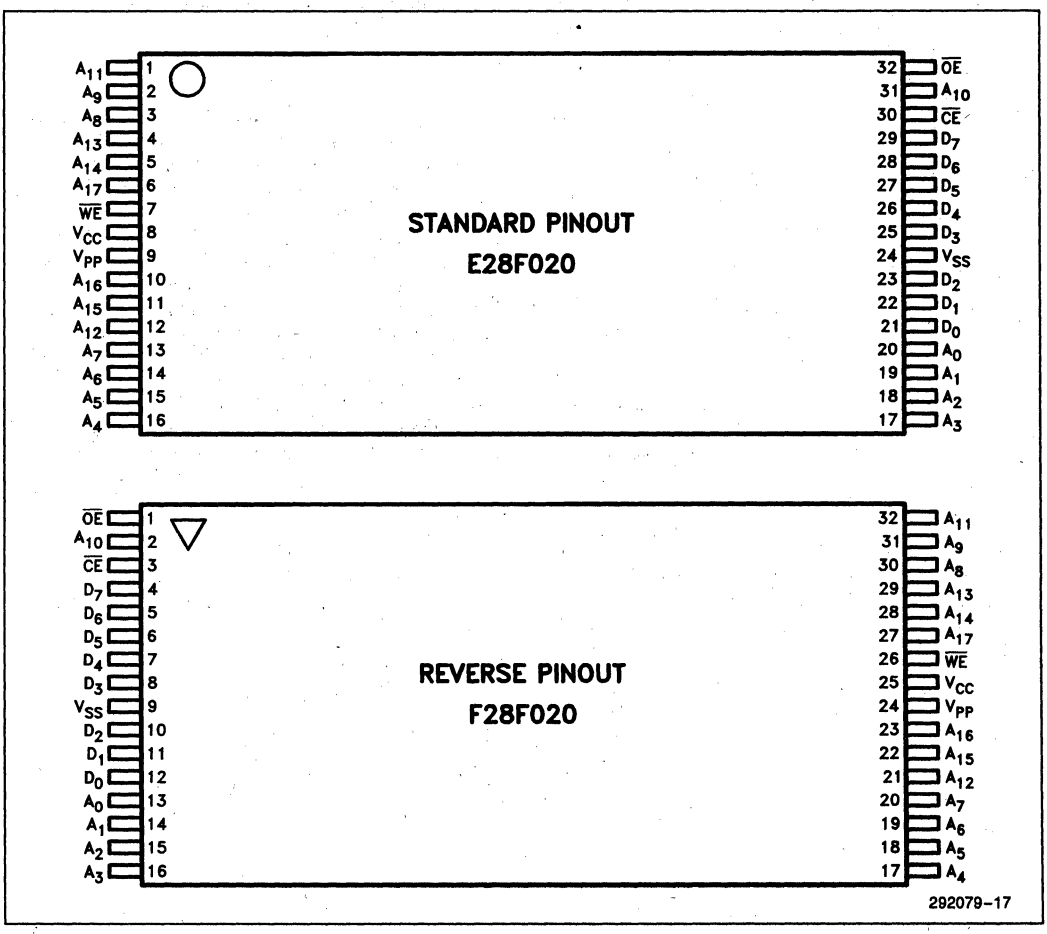


Figure 1. 28F020 32-Lead TSOP—Standard and Reverse Pinouts

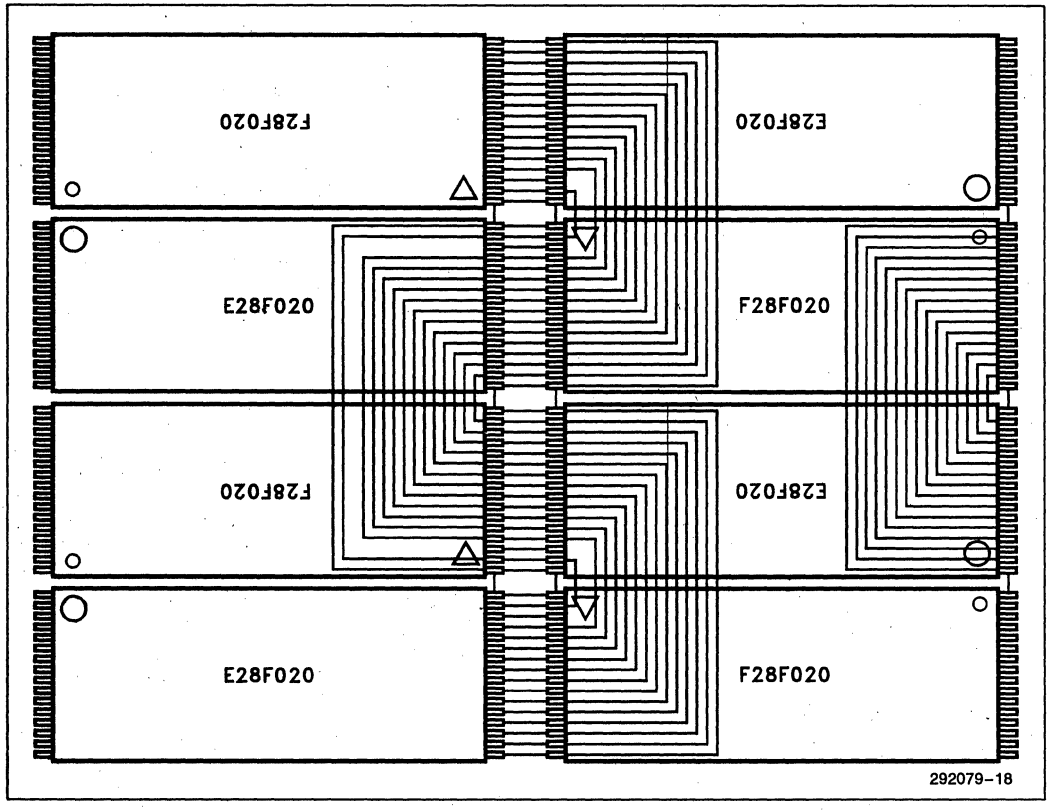


Figure 2. TSOP Optimal Layout: Highest Density Configuration (Conceptual)

Memory Cards

Many laptop and notebook computer manufacturers are pursuing the IC memory card to incorporate a removable mass storage medium. This is an ideal application for the Intel Flash Memory TSOP, due to the package's minimal height.

Solid-State Memory Alternatives

ROM and SRAM are currently the dominant IC card memory technologies. ROM has the advantage of being inexpensive, but is not changeable. When newer software revisions (e.g. Lotus* 123, Wordstar**, etc.) are available, the user must buy a new ROM card for each upgrade. Intel Flash Memory's reprogrammability minimizes the user's expense and the OEM's inventory risk.

SRAM is reprogrammable but batteries are required to maintain data, risking data loss. Like magnetic disks, flash memory is truly nonvolatile and thus has virtually infinite storage time with power off (10 years minimum, 100 years typical). Additionally, SRAM is expensive and not a high density solution. Intel Flash Memory provides a denser, more cost effective and reliable solution.

System level cost is about the same for Intel Flash Memory and SRAM + battery—

Flash memory requires 12V for programming and erasing. If a 12V supply is not available, 5V can easily be boosted. (See Application Note AP-316.) SRAM + battery requires battery state detect circuitry.

Card level cost differences are substantial (Figure 3)—

SRAM must have a battery to retain data. It also requires a V_{CC} monitor and Write Lockout circuitry. Intel's Flash Memory only requires Write Lockout circuitry (switching V_{PP} to 0V is an alternative write protect). This leads to increased area for memory components. More importantly, Intel's Flash Memory density is 4 times that of static RAM, yielding for lower cost per bit.

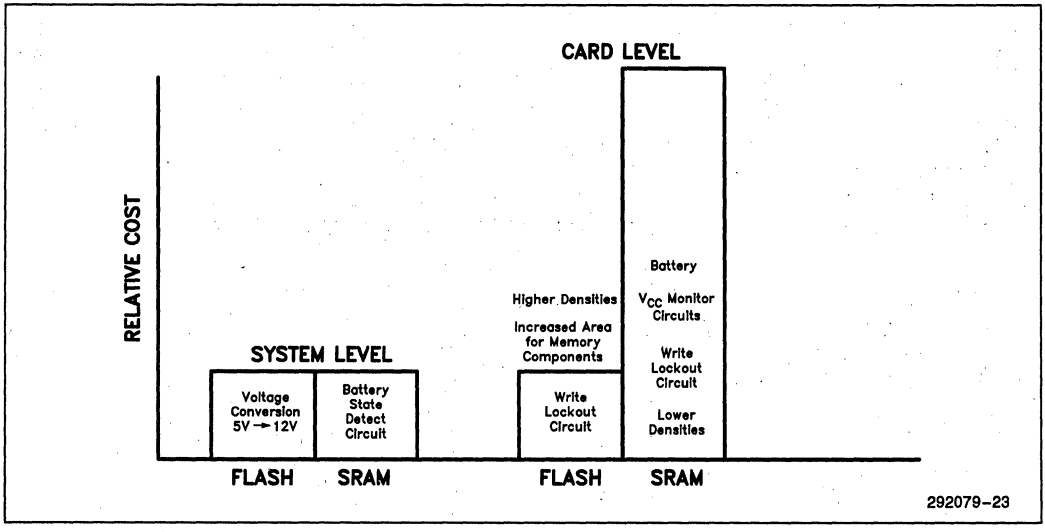


Figure 3. Support Circuitry Cost Comparison

*LOTUS® is a registered trademark of LOTUS Development Corporation.
**WORDSTAR® is a registered trademark of MICROPRO.

Designing a PCMCIA/JEIDA Standard Memory Card

Choosing among IC card design options depends on card architecture (standardization), memory capacity, data bus width, card intelligence, V_{PP} generation, and reliability.

What are the advantages of a standardized memory card pinout?

From the computer system's viewpoint, a standardized pinout enables the use of multiple third-party memory cards. This ensures competitive pricing and wide availability. From the memory card point of view, standardization allows use in a variety of systems.

The Personal Computer Memory Card International Association/Japan Electronic Industry Development Association (PCMCIA/JEIDA) 68-pin format is the emerging IC memory card standard. Several proprietary formats are also available from their respective manufacturers, but these same manufacturers now offer PCMCIA/JEIDA versions. The PCMCIA/JEIDA standard specifies physical, electrical, information structure, and data format characteristics of the card. This standard accommodates either 8- or 16-bit data bus widths.

The following 2 Mbyte memory card design provides a byte-addressable interface using 8-28F020s (2 Mbit, 256k x 8 devices) as shown in Figure 8. While TTL equivalent interfacing is shown, most cards will use gate arrays to reduce chip count. Address lines A18 and A19 are decoded with a 2-to-4 decoder (74HC139) to generate high and low byte chip select signals for each of the 4 pairs of flash memory devices (one pair = high and low byte). The PCMCIA/JEIDA format specifies inputs $\overline{CSL}$ and $\overline{CSH}$ (along with the A0 address line) which select the low and high byte, respectively.

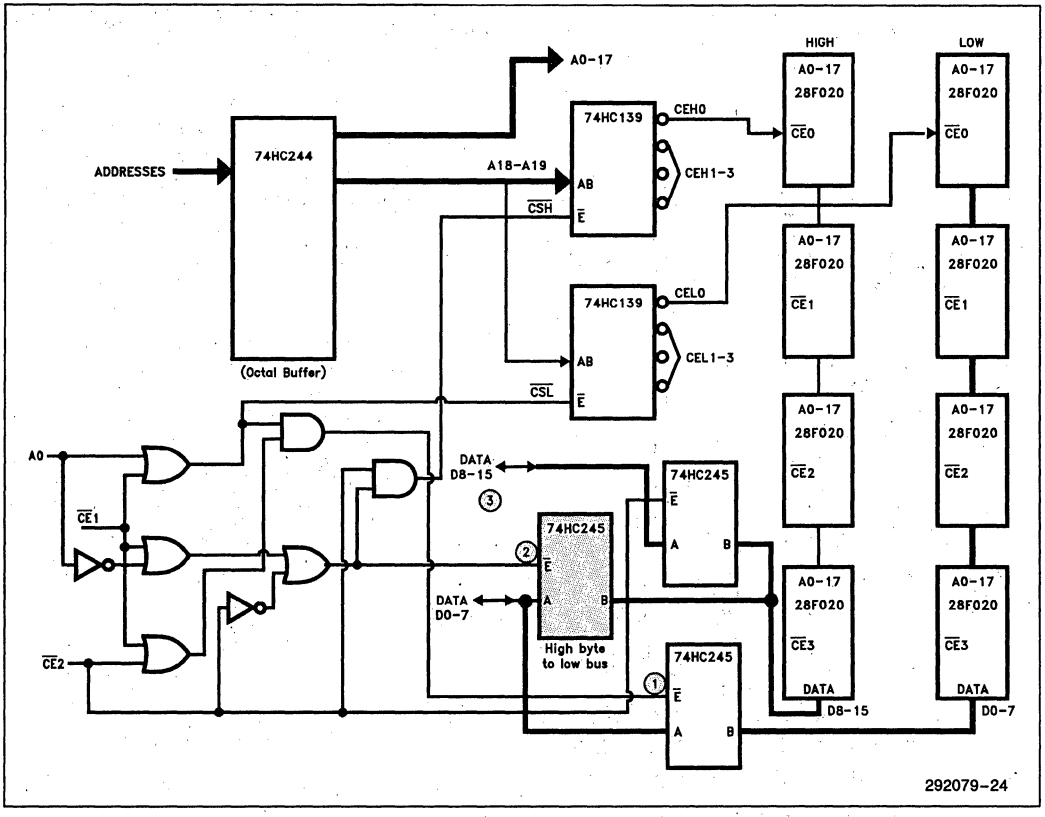


Figure 4. Decoding for PCMCIA/JEIDA Standard Bus Interface

According to the PCMCIA/JEIDA standard, the memory card is designed with the flexibility to have both an 8-bit or a 16-bit interface, dependent upon the machine it is plugged into. When the memory card is plugged into an 8-bit system, the high byte transceiver is multiplexed to the low byte of the system. In Figure 4, the highlighted transceiver (#2), maps the upper byte to the lower byte of the data bus (i.e., D_{8-15} to D_{0-7}). Signals are decoded according to the truth table in the Appendix. (1, 2, and 3 denote transceiver numbers of Figure 8.)

One can double the memory capacity and select from among 8 pairs of flash memory devices by using a 3 to 8 decoder with inputs A_{18-20}. Notice that additional transceivers are not needed to support the additional data fanout. (See section on capacitive loading.)

Single In-Line Memory Module (SIMM)

The SIMM is optimal where minimized board space and upgrade capability are required. Compared to using 8 discrete PLCCs plus capacitors (3019.4mm^2), the equivalent memory capacity SIMM (926.1mm^2) consumes 70% less motherboard real estate.

The SIMM can be built as an 80-pin, 0.050 mil centerline lead spaced, insertable module designed with a 16-bit wide data bus interface. The SIMM pin configuration allows convenient implementation:

- No Address or Data Bus Multiplexing—RAS# and CAS# are not needed;
- Reserved Pins—For product expansion and enhancements: Upgrade capability to 128 Mbytes;
- Presence Detect Eliminates Jumpering—Simplifies user installation.

The 80-pin definition of the flash memory SIMM includes 7 pins for Presence Detect (PD). (See Appendix). The PD pins are read to determine module memory capacity and speed of the devices. The PD pins are either Open circuit or Shorted to ground. By attaching a pull-up resistor to each pin, Open circuits will read as a binary 1 and Shorts as a binary 0. Before implementing the presence detect feature, define your system criteria:

How many modules will be used?

Decide how much total memory your system is to contain. The limit is dictated by the space available, as well as cost.

Flash memory SIMMs can easily accommodate different memory capacities and speeds. Could your system handle mismatched SIMMs?

There are two basic design implementations for interpreting presence detect information. The first approach requires that matching SIMMs are used. The PD pins of all SIMMs are tied to one transceiver that is read as an I/O port (Figure 5).

Invalid reads occur if the user installs mismatched SIMM configurations. Any PD pin shorted to ground makes an open circuit pin appear as a binary zero (0). Mixing module speeds is acceptable, but the PD pins reflect the slower module.

The second approach, allowing any mixture of flash SIMMs, requires more hardware and software for interpretation. The PD pins from each SIMM have separate transceivers, resistors, and I/O ports (Figure 6). Flexibility is increased at the expense of board real estate.

Assume your system accommodates several SIMMs but complete population is not needed. Can the system handle empty sockets?

SIMM upgrade capability is not limited to increases in memory density. A system may be designed with several SIMM sockets on the circuit board. To keep initial end-customer costs down, the system ships with only one SIMM installed. This provides the option of populating the empty sockets at a later time. The PD pins are designed to eliminate jumper or software setups by the end-user when SIMM upgrades are made.

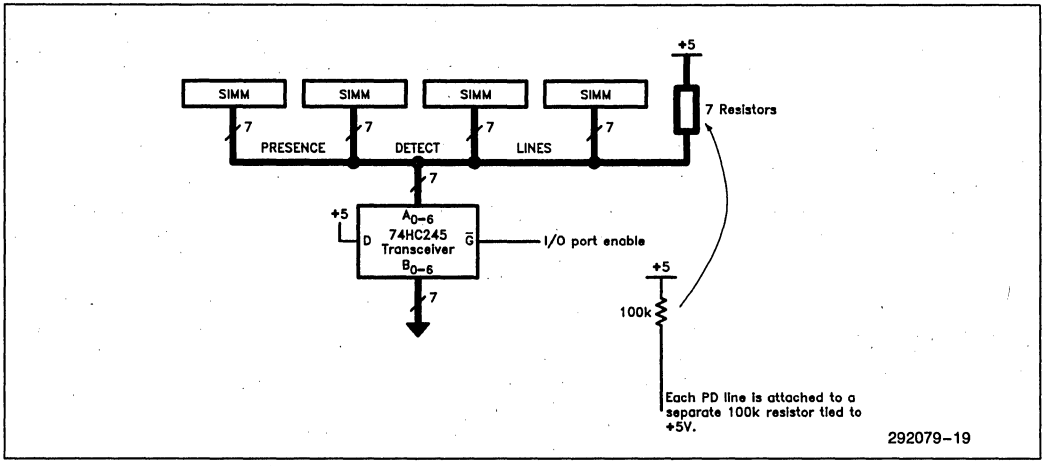


Figure 5. All SIMMs Should Reflect the Same PD Configuration

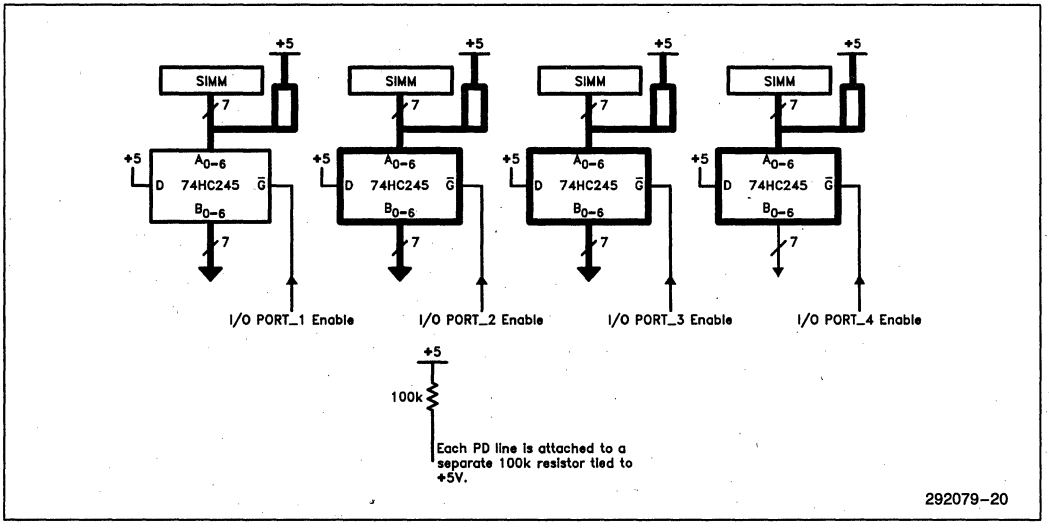


Figure 6. Multiple I/O Transceivers Are Needed if Mismatched SIMMs Are Used

Using the previous scenario, will it matter which socket is used? In other words, what is the installation procedure?

With respect to the PD feature, it does not matter which sockets are full. (However, most designers request that sockets are filled in sequential order to minimize hardware and softwear requirements.) To explain this, look again at the bit-level interpretation of the PD pins. An empty socket also appears as an open circuit. Your software can determine a full (or empty) socket in one of two ways:

Method One (Figure 5)—Reading the PD pins is insufficient. An empty socket will reflect the value of the full socket. Your software will have to read the chip level device identifier hardwired in each flash memory device. (See Intel Flash Memory data sheets regarding inteligent identifiers.) Reading an invalid device identifier from a SIMM address signifies an empty socket. Software demonstrating the use of this method to determine memory capacity is discussed further in the section on "Verifying Paged Memory Board Functionality".

Method Two (Figure 6)—Each SIMM's PD pins are read separately. Reading all ones (the result of all Open circuits) signifies an empty socket. The chip level device identifiers should still be read to establish the number of flash memory devices on the SIMM.

Presence Detect for WAIT-State Interpretation

Using Method One or Method Two from above, the device speed information is read from the pins. This information can be interpreted by software to issue the proper command to the system's programmable WAIT-state generator. By guaranteeing the use of matching SIMMs, the WAIT-state generator would not have to be reprogrammed each time a different SIMM is accessed.

A hardware driver alternative implements an 85C220 EPLD configured with an internal counter (Figure 7). The rising edge of the clock, following Chip Enable going active, latches the count value derived from the PD speed pins (Figure 8).

Each subsequent rising edge of the clock input decrements the counter. A $\overline{\text{READY}}$ signal is output to the CPU (or the system's READY logic circuitry) when the count reaches zero (0). The $\overline{\text{READY}}$ signal remains active until LOAD (Chip Enable, $\overline{\text{CE}}$) goes inactive at the completion of the bus cycle.

The clock signal for the internal EPLD counter is derived directly from the CPU, therefore the count rate and WAIT-states will be system dependent. An EPLD Advanced Design File was generated to demonstrate this application. (See Appendix A.) This is a straightforward approach until designing systems, such as power-saving laptops, that have changeable system clock rates.

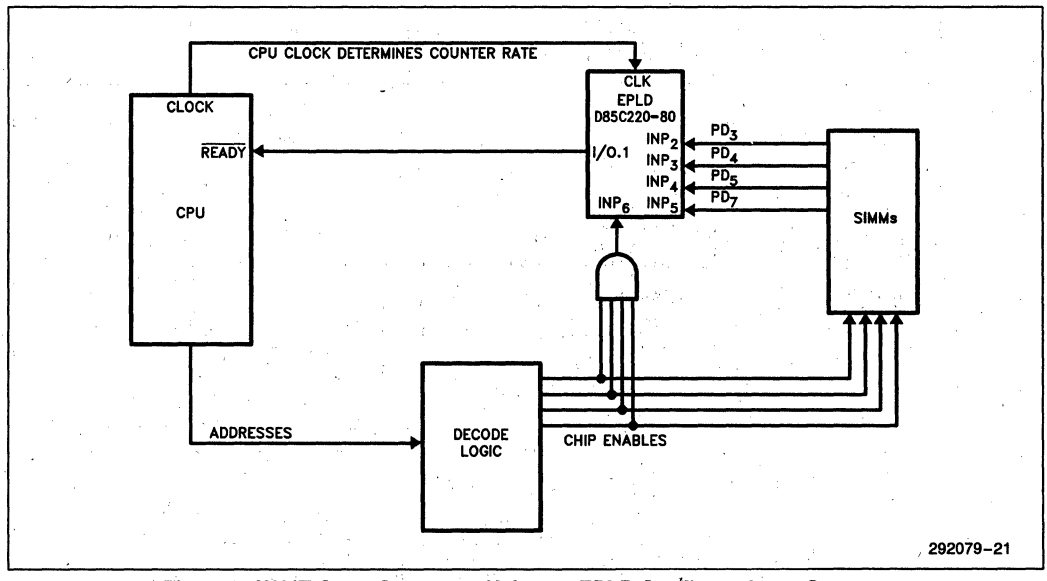


Figure 7. WAIT-State Generator Using an EPLD Configured as a Counter

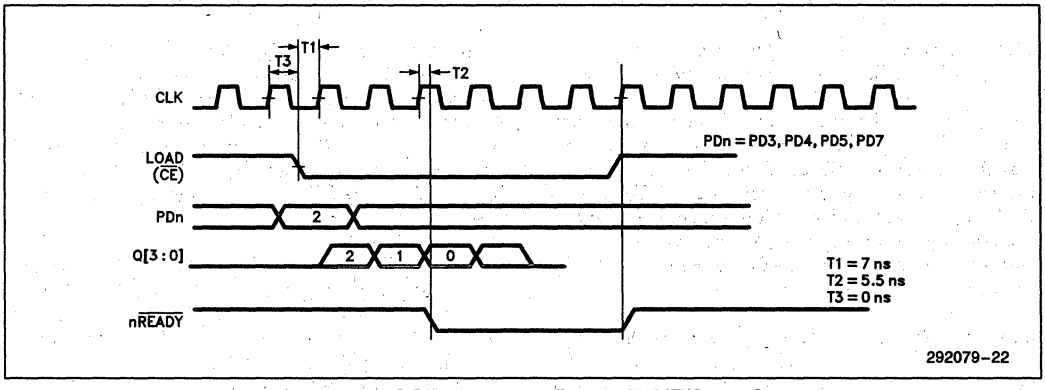


Figure 8. Timing for SIMM Presence Detect WAIT-State Generator

HARDWARE DESIGN IMPLEMENTATIONS

Paged, linear, and I/O are the three fundamental addressing methods that can be used for accessing an array of memory devices. Linear addressing offers the fastest and most direct access to a memory array. It consumes the largest portion of the system's memory and is only practical in a 386™ microprocessor (or other 32-bit processor) family system because of the large memory space available above 1 Mbyte. The I/O mapped memory array consumes the smallest amount of the system address space but has the lowest performance. A page-mapped memory array, also called a sliding AT window, is a hybrid of the linear and I/O designs. The memory array is usually very large relative to the system interface, consisting of pages typically ranging in size from 8 Kbytes to 64 Kbytes. (LIM-EMS use four to twelve 16 Kbyte pages.)

Design Example—A Paged-Mapped Memory Board

A paged design employs addressing techniques similar to the Lotus-Intel-Microsoft expanded memory specification (LIM-EMS). It allows access to one or more sections (or pages) of the flash memory array at a time. This minimal interface is particularly useful within the DOS 1 Mbyte memory space. The DOS map (Figure 9) shows 128 Kbytes of memory space available in the Optional I/O Adapter ROM area. LIM-EMS, LAN, the flash memory design discussed in the following sections, and other accessory cards can use this area.

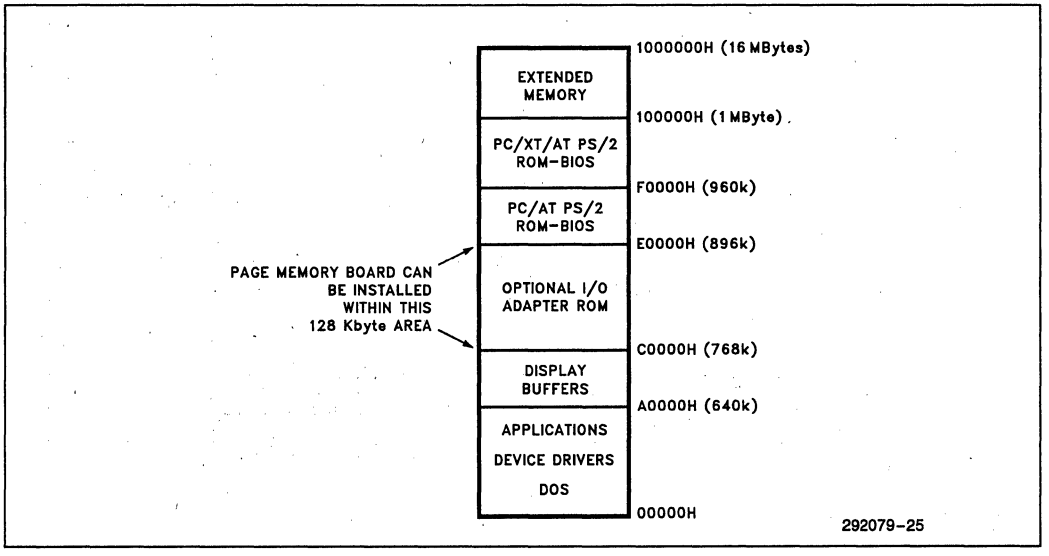


Figure 9. DOS Memory Map

Figure 10 shows the block diagram of the page-mapped flash memory board design. (Except for the addressing method, all the functional components of this board could be used on a linear or I/O mapped flash memory array.) This PC-AT*** compatible design example consists of a flash memory array (using SIMMs) and the corresponding memory and I/O decoding, V_{PP} generation, and the interface to the system bus. (Component numbers shown with the following diagrams correlate with the actual schematics in the appendix.) A page size of 64 Kbytes is used. Depending on the system's configuration, memory contention may require a smaller page size. (Note that the LIM EMS 4.0 standard uses 4 contiguous 16 Kbyte pages. Multiple pages can exist as space permits.)

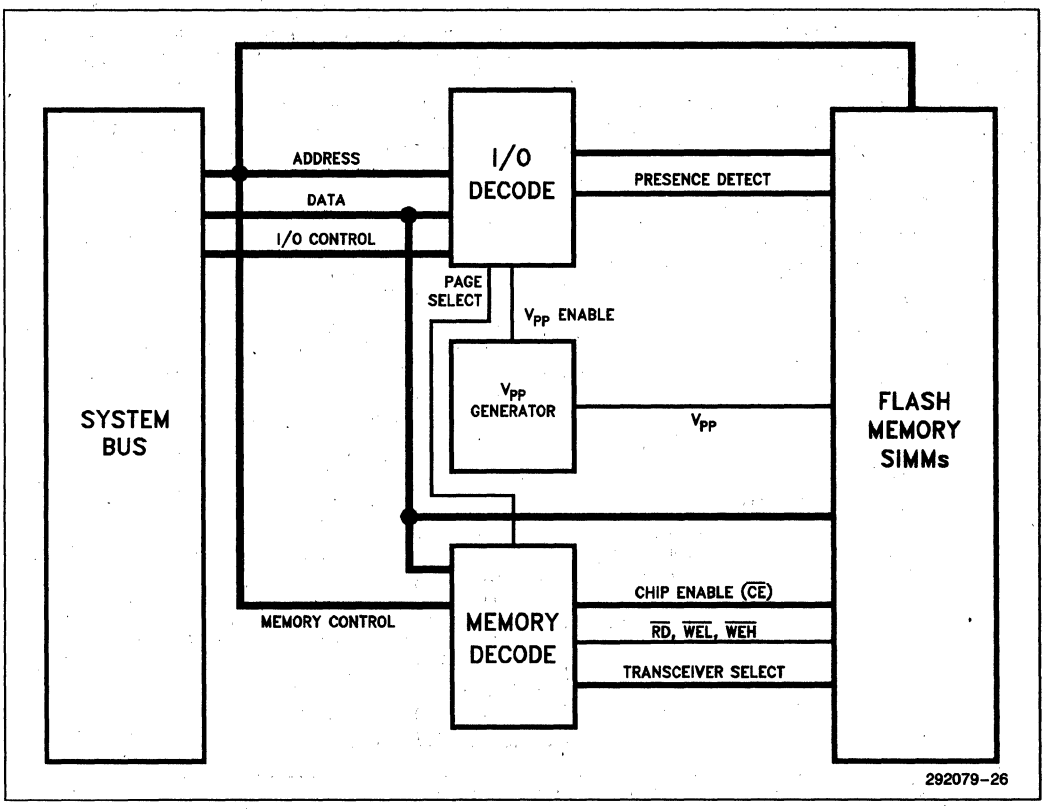


Figure 10. Page-Mapped Flash Memory Board

NOTE:
A similar hardware platform using Intel Flash Memory Cards is contained and described in the Hardware Reference Manual that can be ordered separately through Intel literature (Order Number 296871).

***PC-AT® is a registered trademark of International Business Machine Corporation.

The Decoding Scheme

The Intel Flash Memory on this board is installed in 4 SIMM sockets. With a fully populated board, the memory capacity ranges from 4 Mbytes to 16 Mbytes depending on the SIMM density used.

Depending on the density, up to eight chip enables, $\overline{CE}_0$–$\overline{CE}_7$, are used on a SIMM (4 $\overline{CE}$s for 8-chip, 8 $\overline{CE}$s for 16-chip SIMMs). Standard decoding techniques generate separate chip enables, output enables, and write enables. This method has the disadvantage of having to accommodate a large number of traces. The addressing scheme incorporated in this design minimizes the number of board traces neeed to select individual devices. Device selection is made on a row-column basis where: rows are Output Enables ($\overline{OE}$s), Write Lows ($\overline{WRL}$s), and Write Highs ($\overline{WRH}$s) and columns are Chip Enables ($\overline{CE}$s). (For low-powered systems, this method may be unacceptable because each chip enable activates a maximum of 8 components.) These signals are generated by decoding the page lines P_3–P_7 (Figure 11, U22). (See Page Number section.) Pages within a component are selected by tying P_0, P_1 and P_2, respectively, into pins 37 (A_{15}), 36 (A_{16}), and 35 (A_{17}) on the SIMM (Pin 35 is a no-connect (NC)).

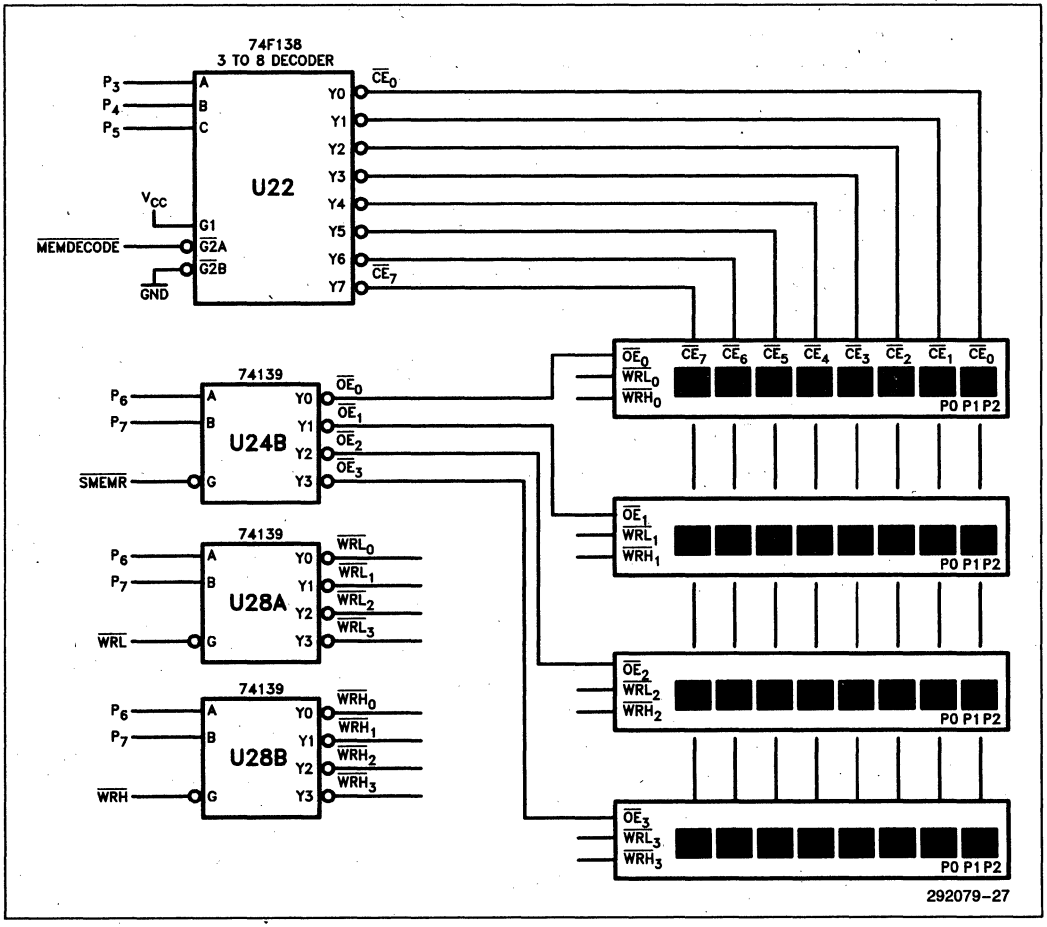


Figure 11. Individual Components Selected by Row-Column Addressing Saves Board Traces

Planning for upgrades also presents another interesting situation. The 1 Mbyte SIMM has four chip enables ($\overline{CE_0}$–$\overline{CE_3}$), one for each pair of components. The pair of components represent high and low bytes and are selected by $\overline{WRH}$ and $\overline{WRL}$, respectively.

The 1 Mbyte SIMM represents sixteen 64-Kbyte pages (eight 128-Kbyte components). To accommodate upgrade capability, pages will not be contiguous because a "4-page hole" exists every 4 pages (P2 is attached to a no connect). To overcome this rearrange the page select lines with jumpers (Figure 12):

1 Mbyte	SIMM	(8 * 28F010s)	JP2, JP4, JP6, JP8
2 Mbyte	SIMM	(16 * 28F010s)	JP2, JP3, JP7, JP9
2 Mbyte	SIMM	(8 * 28F020s)	JP1, JP3, JP5, JP7
4 Mbyte	SIMM	(16 * 28F020s)	JP2, JP3, JP6, JP7

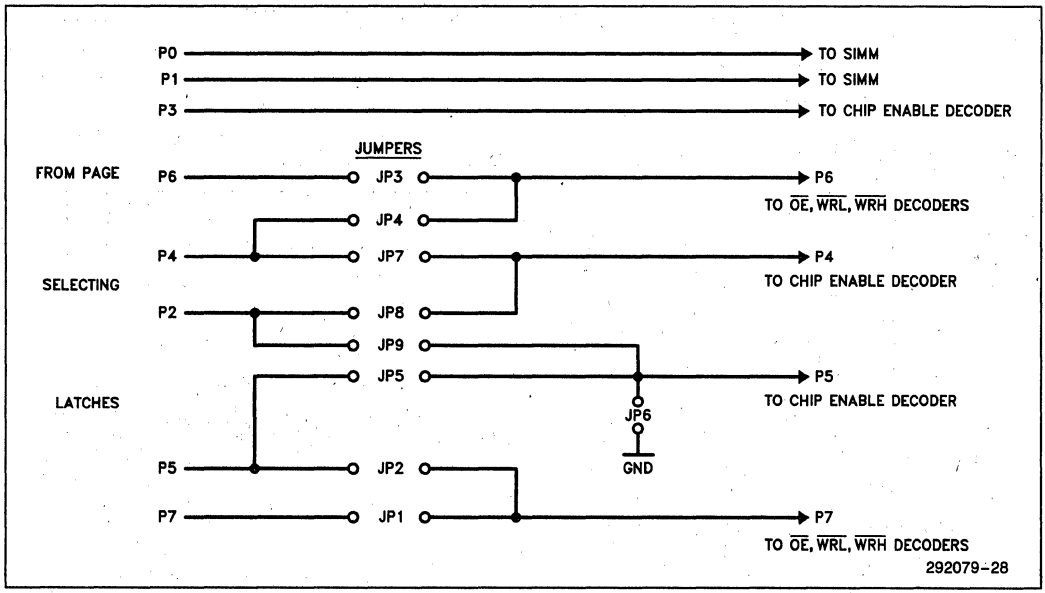


Figure 12. SIMM Density Upgrading Jumpers

If a system is designed that uses PCMCIA/JEIDA standard memory cards instead of SIMMs, this decoding is greatly simplified. The memory card is treated like a large memory array. Using a 64 Kbyte page size as an example:

Address lines A_{0-15} are supplied directly from the system address bus (after buffering). Address lines A_{16-23}, which select the pages, are sent as data to a latch before entering the memory card (Figure 13).

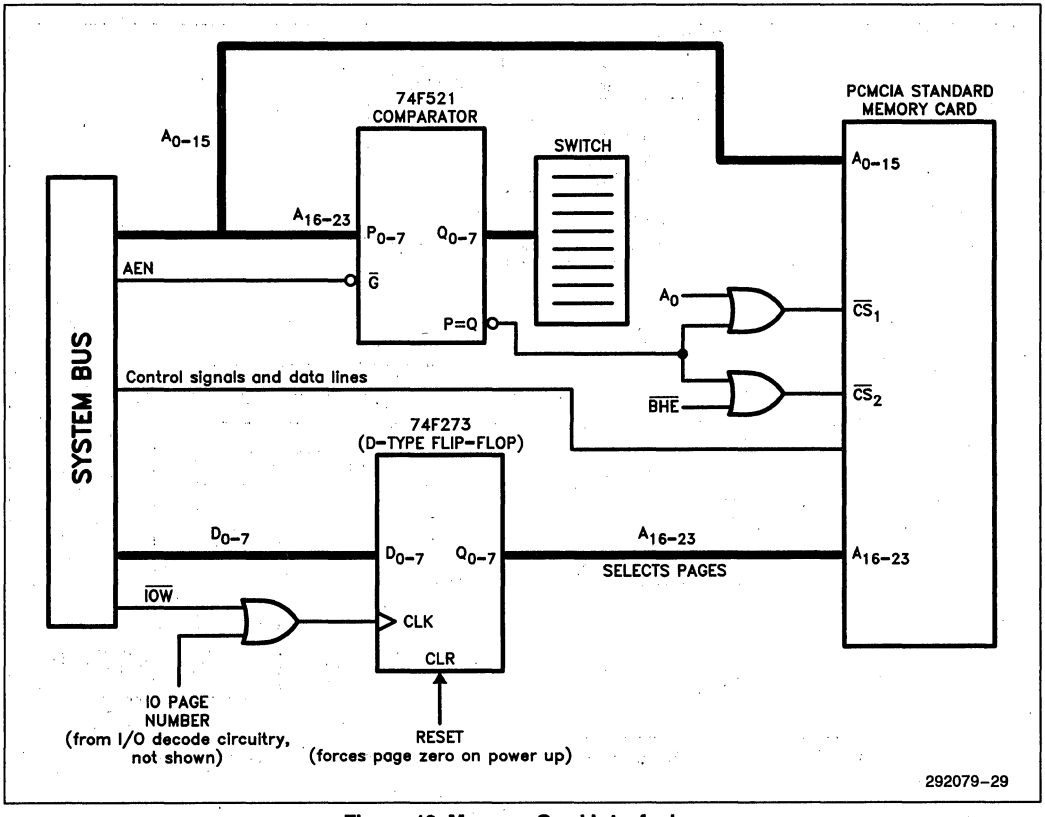


Figure 13. Memory Card Interfacing

The page inputs to the "Chip Enable" decoder (Figure 14, U22) are redefined as follows:

P3 = P3, P4 = P2, P5 = GND,

P6 = P4, and P7 = P5.

For a better understanding, you should verify the bit combinations while stepping through the first few pages. Notice that the sequencing of page numbers does not correspond linearly with the Chip Enables. This is not significant because the data is read the same way it is written.

Page Numbers	Chip Enable
0, 1, 2, 3	0
4, 5, 6, 7	2
8, 9, 10, 11	1
12, 13, 14, 15	3

NOTE:
Linear Page Selection Results in Nonlinear Chip Enable.

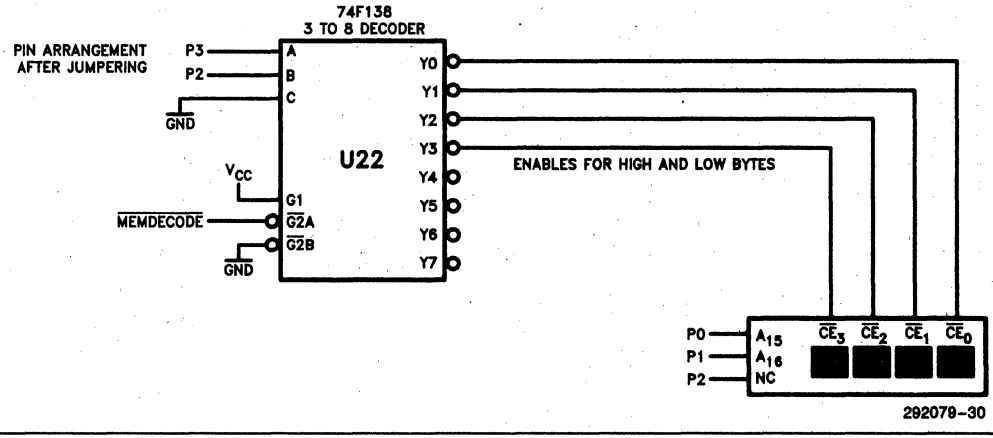
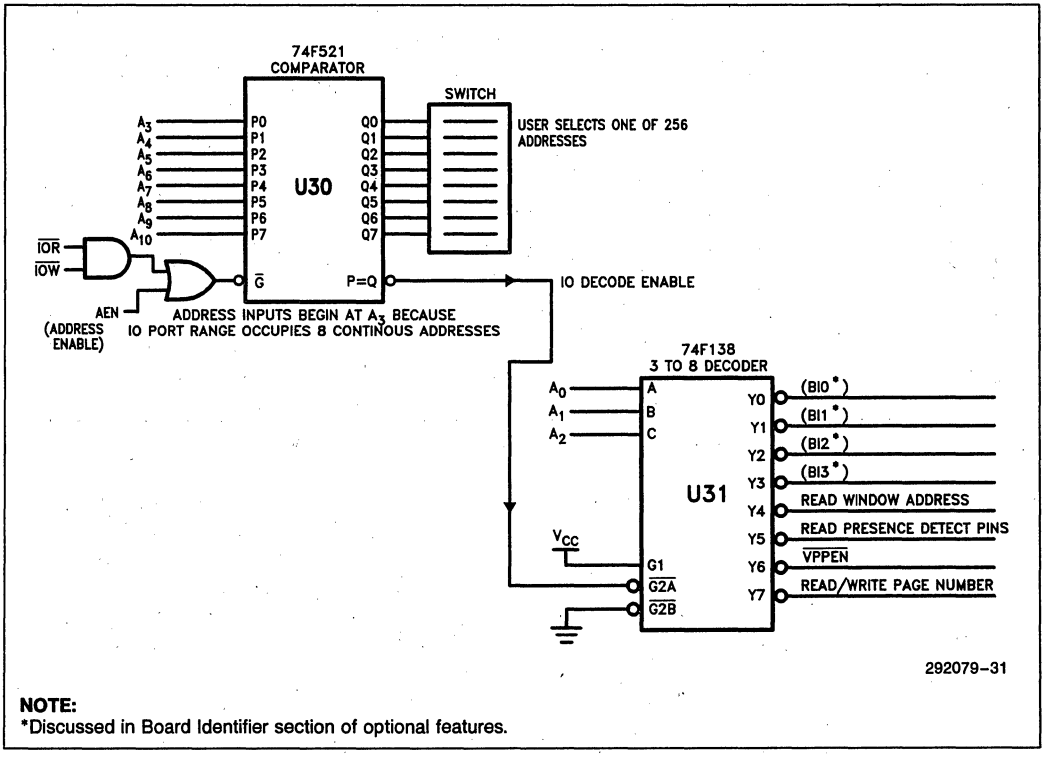


Figure 14. Component Selection Relative to Page Number for 1 Mbyte SIMM

I/O Decoding

Multiple functions can be implemented with I/O decoding access. Some examples include: reading the current window address, reading the presence detect pins, enabling V_{PP}, and reading/writing the page number. The eight consecutively addressed I/O ports on this board (4 reserved for optional features) are located at a user-selectable address. This base I/O port address is setup on an 8-byte boundary by using A_{3-10} as inputs to the 74F521 comparator (Figure 15, U30). When any of the eight consecutive I/O port addresses matches the dip switch settings (and AEN is low), the comparator outputs the I/O Decode Enable (to decoder U31).

AEN (address enable), the chip select for the 74F521 comparator, is supplied by the PC I/O channel. It distinguishes processor bus cycles from DMA bus cycles. A high on AEN indicates that a DMA (or DRAM refresh) cycle is in progress and we must stay off the bus. The enables for the 74F138 IODECODER (U31) are provided by IODECODE ENABLE along with the "ANDing" of $\overline{IOR}$ and $\overline{IOW}$. This decoder selects the I/O ports that access the page window address, the SIMM presence detect pins, the V_{PP} Enable, and the page number. Each of these I/O ports are described in detail:



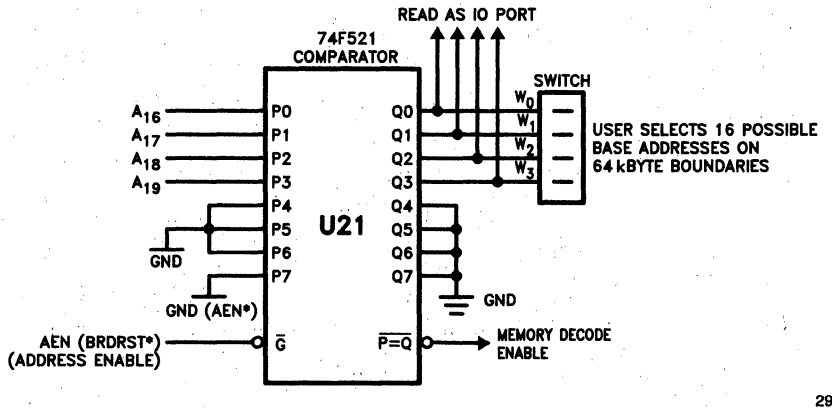
NOTE:
*Discussed in Board Identifier section of optional features.

Figure 15. User Selectable I/O Base Address for I/O Decoding

The Window Address

The user-selectable window address can be set up on any 64K boundary below 1 Mbyte. (The memory window should be placed between C0000h and E0000h to be DOS compatible.) A DIP switch (connected to a transceiver for reading) and the four address lines A_{16-19} are the inputs to the 74F521 comparator (Figure 16, U21). There are 16 possible window addresses. The comparator outputs the "Memory Decode Enable" signal when an address is selected that is within the 64 Kbyte window. This signal (with AEN low) allows board level memory decode.

The location of this 64 Kbyte window can be moved above 1 Mbyte by adding A_{20-23} to the comparator's inputs P_4 to P_7 of the 74F521. Bits D_{4-7} of the data bus can be connected to the comparator's pins Q_4 to Q_7 to allow reading of the full base memory address.



*NOTE:
Ignore the contents within parentheses. This is used for section on master/slave configuration.

Figure 16. User Selects Base Memory Address

Presence Detect

The method shown earlier in Figure 3, is used to configure the PD pins in this design. SIMMs can be added incrementally only in similar densities. The SIMM PD pins are read by selecting the appropriate I/O address that enables the 74F245 transceiver.

V_{PP} Generation

V_{PP} is generated locally (on this board) to ensure a stable, switchable 12V (±5%) supply. (Many systems generate their own 12V power supply. However, it should not be used if its regulation is greater than 5%.) On power-up, system reset, or when V_{CC} is below 4.5V, V_{PP} is forced off. It is enabled (or disabled) by writing to the I/O port address (Figure 15, U31) that generates the $\overline{VPPEN}$ signal. This on/off capability is essential for battery-operated equipment and eliminates the need for $\overline{WE}$ filtering (as discussed below). (See Intel data sheet for V_{PP} standby current.) The $\overline{VPPEN}$ signal "ORed" with the system I/O write, $\overline{IOW}$, functions as the clock signal for the 74HC74 D-flip flop (Figure 17, U42A). The D-input is latched when $\overline{IOW}$ goes high. Writing a one or a zero turns V_{PP} on or off, respectively.

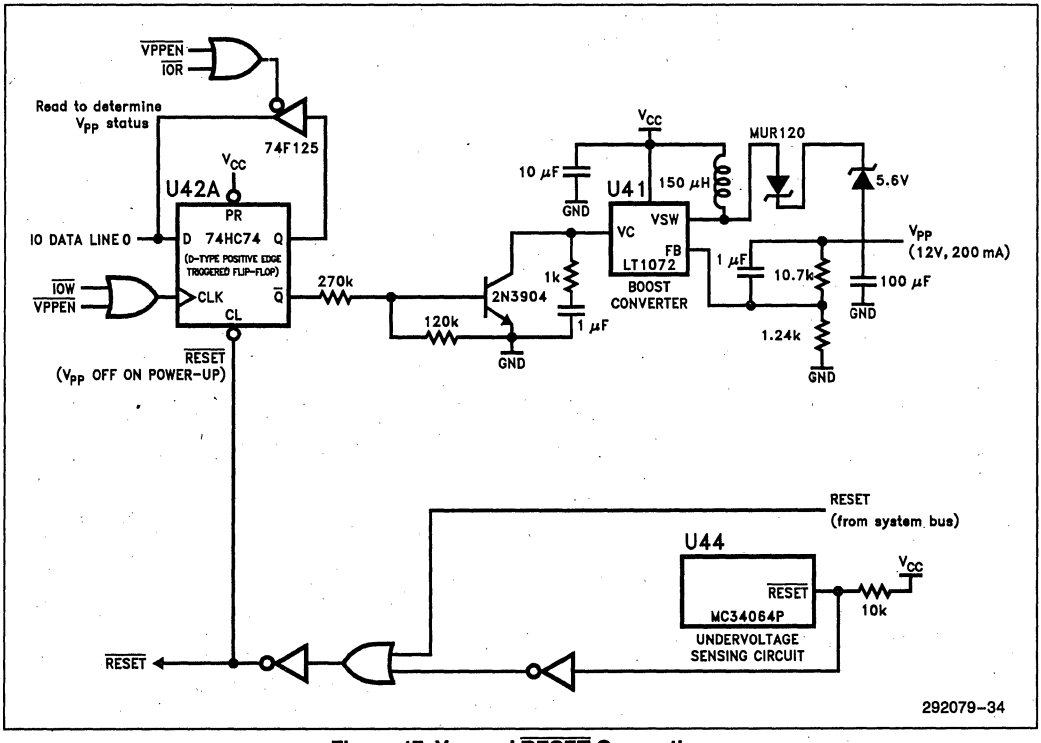
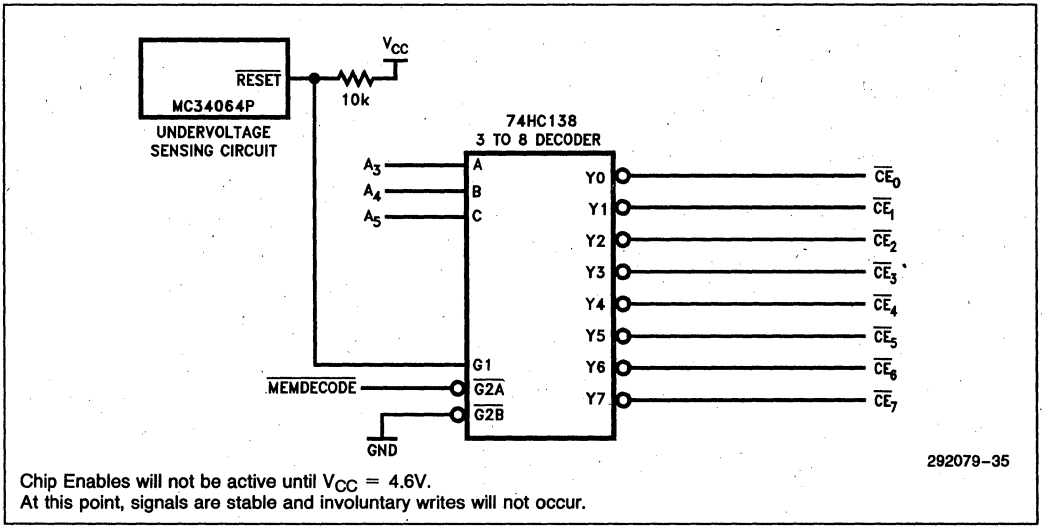


Figure 17. V_{PP} and $\overline{RESET}$ Generation

Linear Technology's LT1072 (U41) switching regulator is used as a 5V to 12V boost converter. The FB input regulates the voltage output. The 10.7k and 1.24k resistors are used to establish the correct reference voltage to obtain 12V. The 100 μF capacitor at the output is used to handle up to 200 mA. (See Linear Technology's LT1072 data sheet for more information.) Typically this will be much more than needed and a smaller capacitor can be used. However, this will accommodate interleaving of 8 components but may not be practical in a battery-operated system. (See section on Interleaving in the Software Design Implementation chapter.) Additionally, sufficient time should be allowed when switching V_{PP} on. The delay is a factor of the load on the line and the quality of the passive components chosen. The diode, MUR120, keeps the inductor from absorbing current from the charged output capacitor. The 5.6V zener diode ensures that when V_{PP} is less than 5.6V, the V_{PP} output is held at 0V. (This is optional if $V_{PP} \leq$ 5V is tolerable.)

During system power-up, some probability exists that noise may generate spurious writes which are actually the sequence of flash memory commands that initiate erasure or programming. Power-up protection in this design is provided by disabling V_{PP} until voltages have stabilized. The Motorola component, MC34064P (U44), is an undervoltage sensing circuit that begins functioning when V_{CC} is above 1V. Between 1V and 4.6V, the $\overline{\text{RESET}}$ output is active. The $\overline{\text{RESET}}$ output or a system RESET clears the 74HC74 (U42A), keeping V_{PP} off when V_{CC} is less than 4.6V. Alternatively, this signal, or a supply's "POWERGOOD" signal, may gate $\overline{\text{WE}}$ or $\overline{\text{CE}}$, as is common with battery-backed SRAM or EEPROM designs. As an example, the $\overline{\text{RESET}}$ output of the MC34064P can be tied to the active-high enable of the decoder to disable any $\overline{\text{CE}}$s until V_{CC} = 4.6V, as shown in Figure 18.



Chip Enables will not be active until V_{CC} = 4.6V.
At this point, signals are stable and involuntary writes will not occur.

Figure 18. Protecting the Circuit from Involuntary Erasure and Programming. Use an Undervoltage Sensing Circuit, or a System's "POWERGOOD" Signal, to Control Chip Enables

How is V_{PP} Switched on (Refer to Figure 17):

Latching a one into the 74HC74 D-input (U42A) puts a zero on the output $\overline{Q}$. This turns off the transistor 2N3904. When the 2N3904 is off, the VC input of the LT1072 (U41) is 5V and the VOLTAGE SWITCH (VSW) output generates 12V.

Page Number Selection and Reading

It is standard practice to use an I/O port to generate the page number for this type of memory array. The potential number of pages that can be selected is determined by the size of the data bus as well as the amount of decoding the system can practically handle. In this design, this I/O port allows selection of 256 64-Kbyte pages, for a total of 16 Mbytes of flash memory. The page number is written to the 74F273, Octal D-Type Flip-Flop (Figure 19, U37). It is latched by the rising edge clock signal derived by the "ORing" of the corresponding 74F138 decode signal (I/O PAGE NUMBER) and the system $\overline{IOW}$.

Page zero is automatically selected on power-up because the 74F273 clear input is connected to $\overline{RESET}$ (generated as part of the V_{PP} circuitry). This feature ensures that the board will power up in page zero. Given the proper software, this board can be turned into the system's bootable drive. (See section on Software Design Implementations.)

The current page number can be obtained by reading the same I/O port. The I/O decoder output, I/O PAGE NUMBER, "ORed" with the system $\overline{IOR}$, produces the signal enabling the 74F245 bus transceiver (that is tied to the output of the 74F273).

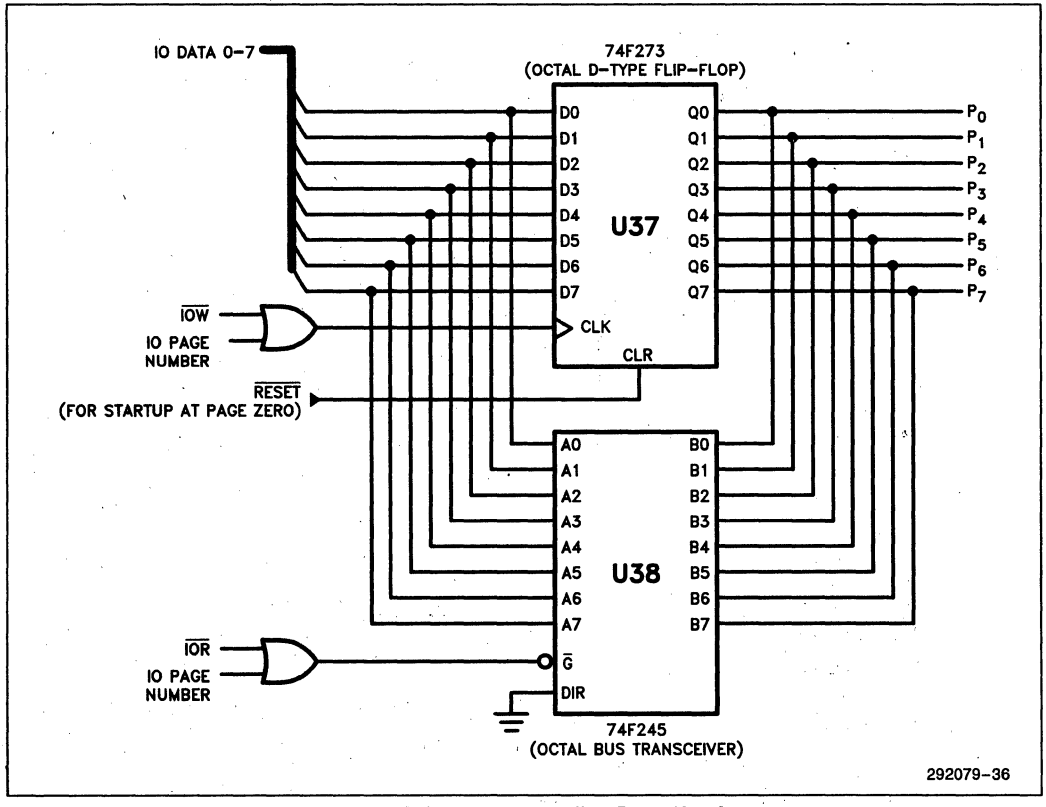


Figure 19. Selecting or Reading Page Number

Design Considerations

The SIMMs high and low bytes are enabled by $\overline{WEH}$ and $\overline{WEL}$, respectively. Using a high and low byte transceiver for each SIMM limits the capacitive loading and prevents performance degradation of the data bus. (This becomes important when upgrading to flash memory SIMMs that have 16 components. See section on capacitive loading.) Also, the PC I/O channel bus specification requires that no more than 2 TTL loads be present on any one line. Therefore, the SIMM transceivers must be routed through two additional transceivers at the PC bus interface (refer to "Switchable Data Bus Width" section). In this paged memory board design, the SIMM transceivers are enabled by a 2 to 4 decoder which uses page pins P_6 and P_7 as decode signals. The enable for the decoder is supplied by the $\overline{MEMDECODE}$ signal; transceivers are disabled unless an address within the 64 Kbyte page is accessed.

Optional Board Features

So far we have described the components required to design a functional flash memory array. Optional features can be added to make this board more versatile in an application environment:

Switchable Data Bus Width

This feature allows the board to execute in a PC XT* (8-bit bus) or a PC AT system (16-bit bus). Memory card designs for adopting the PCMCIA/JEIDA format must include similar provisions as shown earlier. At the PC-I/O channel interface, (for use in an 8-bit system), an extra transceiver is used to redirect the upper data bus (D_{8-15}) to the lower data bus (Figure 20, U9). The $\overline{16BIT}$ signal is generated from a ground on the PC AT I/O channel extension; it will be high (because of the pull-up resistor) when a PC XT is used. (The $\overline{16BIT}$ signal can be read by software through the 8th bit of the Presence Detect port.)

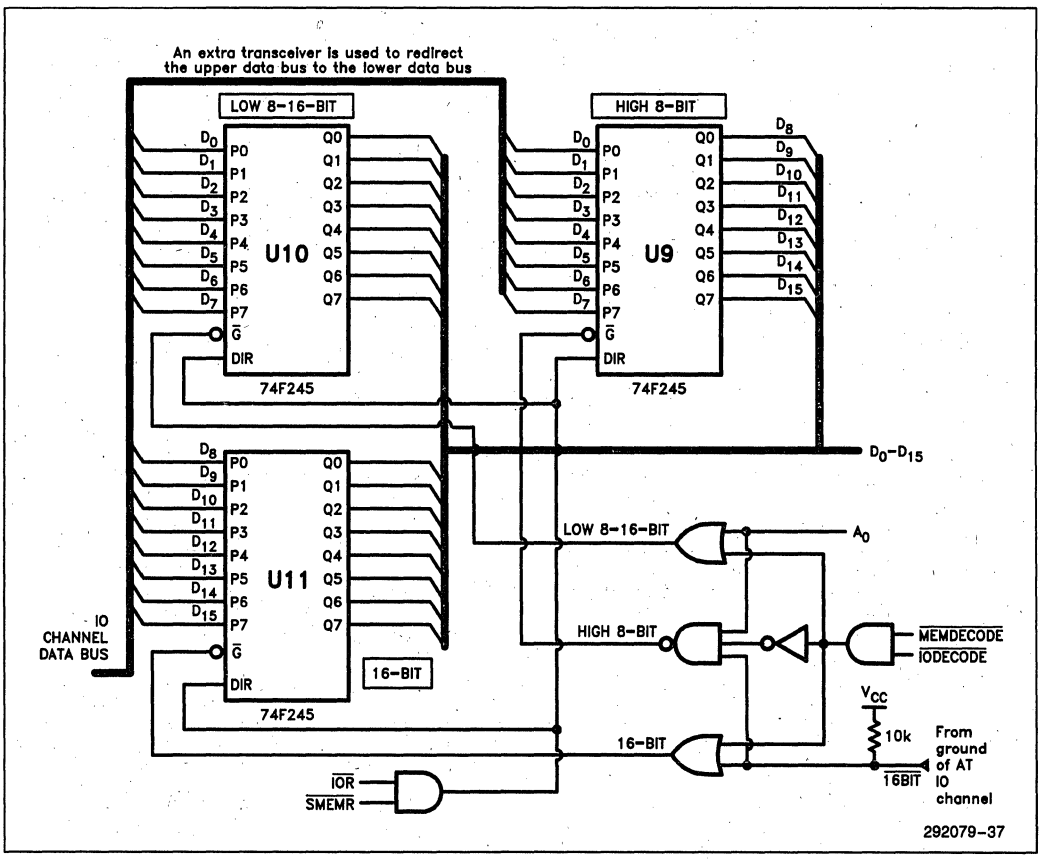


Figure 20. I/O Channel Transceiver Interface for 8- or 16-Bit Data Bus Selection

*PCXT® is a registered trademark of International Business Machine Corporation.

Access to a word (2 bytes) requires two bus cycles to generate two addresses in an 8-bit system. As an example referring to Figure 20, when accessing a memory word at address zero (0):

$\overline{16BIT}$ = 1, $\overline{MEMDECODE}$ = 0;

During access to the low byte → A0 = 0, so the signal "LOW 8/16 BIT" is active;

During access to the high byte → A0 = 1, so the signal "HIGH 8 BIT" is active.

The high byte from the SIMM is multiplexed onto the low byte of the system bus.

The circuitry at the SIMM transceiver interface determines whether to use the Bus High Enable ($\overline{SBHE}$) signal or A_0 to select the high byte. The $\overline{16BIT}$ signal selects the "A" or "B" inputs of the 74F157 multiplexer (Figure 21, U27). Regardless of the bus size, the $\overline{WRL}$ signal is generated on a system memory write ($\overline{SMEMW}$) to an even address (A0 = 0). During a 16-bit write, the $\overline{WRH}$ signal is generated by a system memory write to the high bus ($\overline{BHE}$). However in an 8-bit system, where $\overline{SBHE}$ is absent, the $\overline{WRH}$ signal is generated by a system memory write to an odd addressed byte (A_0 = 1).

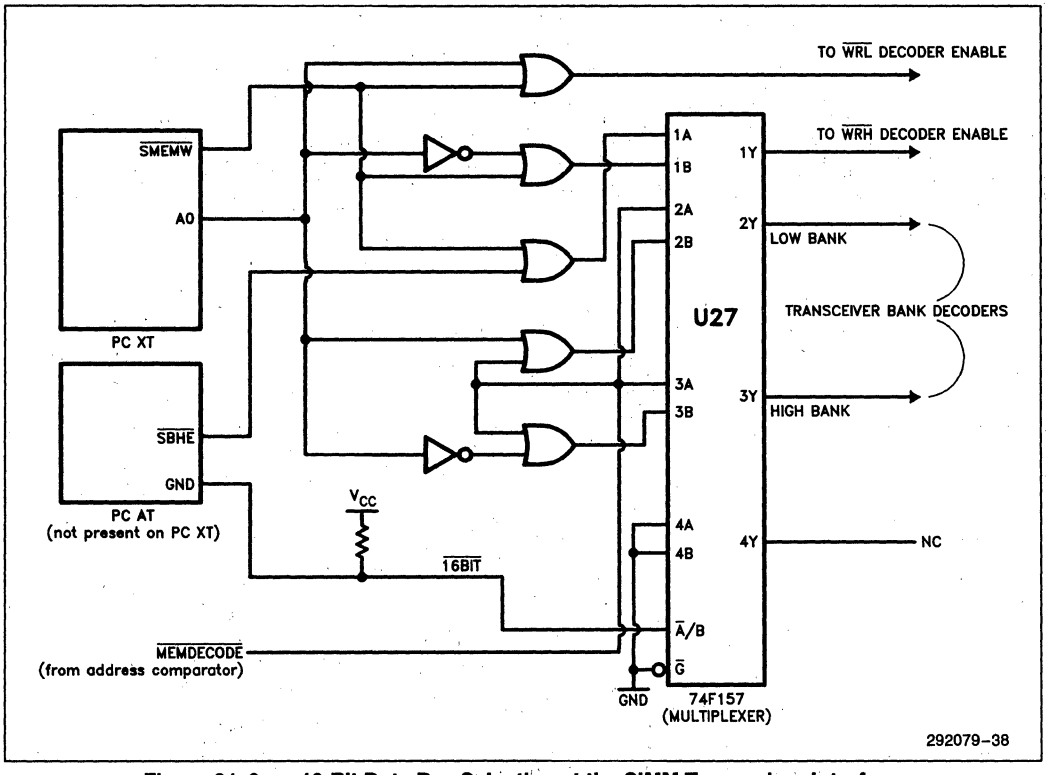


Figure 21. 8- or 16-Bit Data Bus Selection at the SIMM Transceiver Interface

The eight transceivers for the four SIMMs are selected by signals T_{0-7}. Even (T_0, T_2, T_4, T_6) and odd (T_1, T_3, T_5, T_7) numbered signals decode for the SIMM low and high bytes, respectively. The signals T_{0-7} are derived by decoding P_6 and P_7 (Figure 22, U24A) and the transceiver bank decoders (Figure 21, U27).

For a 16-bit system, the $\overline{\text{MEMDECODE}}$ signal selects both the low and high banks. For an 8-bit system, the low bank is selected by generating an even address (A_0 = 0) in conjunction with the $\overline{\text{MEMDECODE}}$ signal. Since $\overline{\text{SBHE}}$ is absent (in an 8-bit system), the high bank is selected by an odd address (A_0 = 1) in conjunction with the $\overline{\text{MEMDECODE}}$ signal.

Master/Slave Configuration

This feature allows the system to accommodate more than one board. The board reset signal, $\overline{\text{BRDRST}}$, of Figure 23 is used to enable the board. The comparator (Figure 16, U21) that generates the MEMORY DECODE ENABLE must be reconfigured:

1. AEN is connected to P_7;
2. $\overline{\text{BRDRST}}$ is connected to the chip enable, $\overline{G}$.

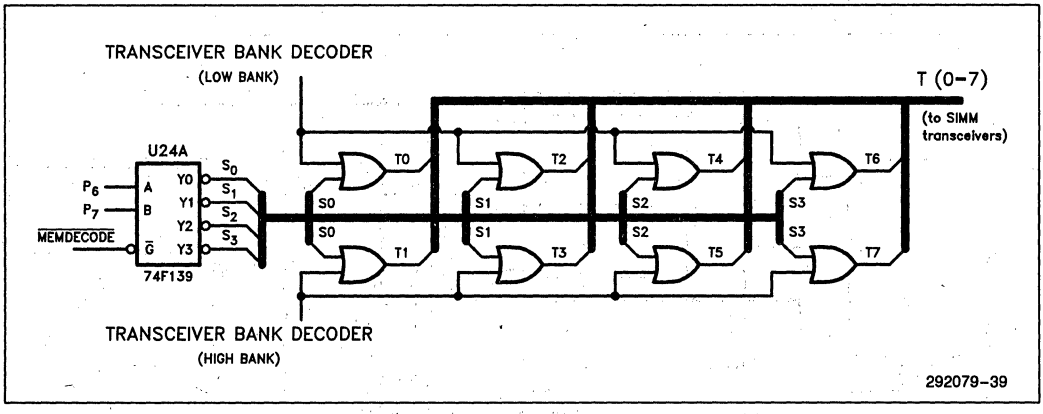


Figure 22. Transceiver Selection at the SIMM Interface

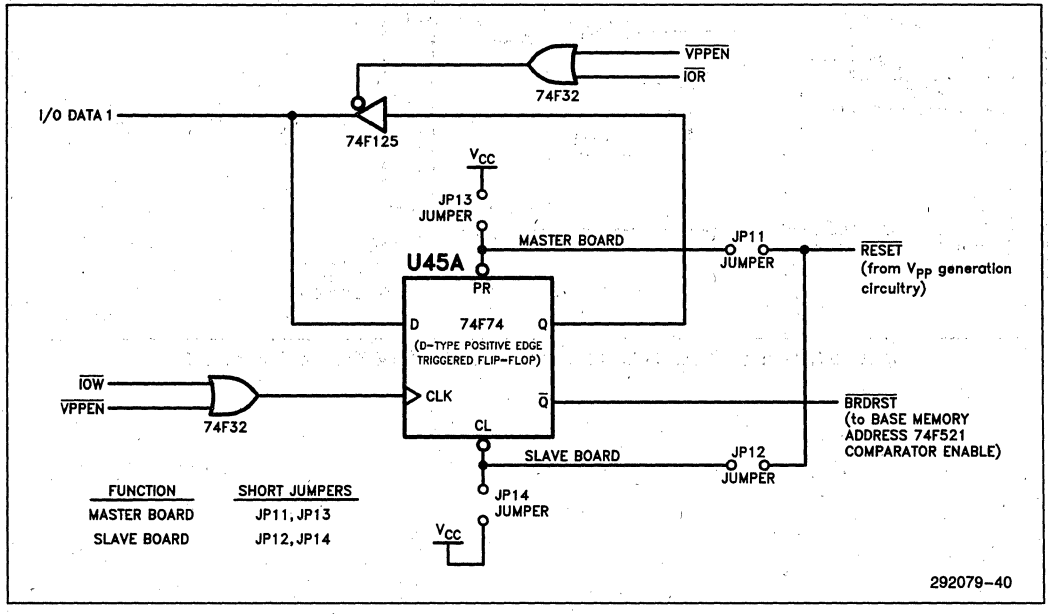


Figure 23. Master/Slave Configuration for Multiple Boards

The jumper settings determine if the board is "active" on system reset ($\overline{BRDRST}$ will be low). The Master/Slave port is shared with V_{PP} enble; therefore to change the "active" status of the board, write to the $\overline{VPPEN}$ I/O port. Software should first read this port to determine the status of "V_{PP} Enable", then use the appropriate mask technique to activate or deactivate the board.

Board Identifier

The board identifier, occupying 4 additional I/O ports, is used for two functions:

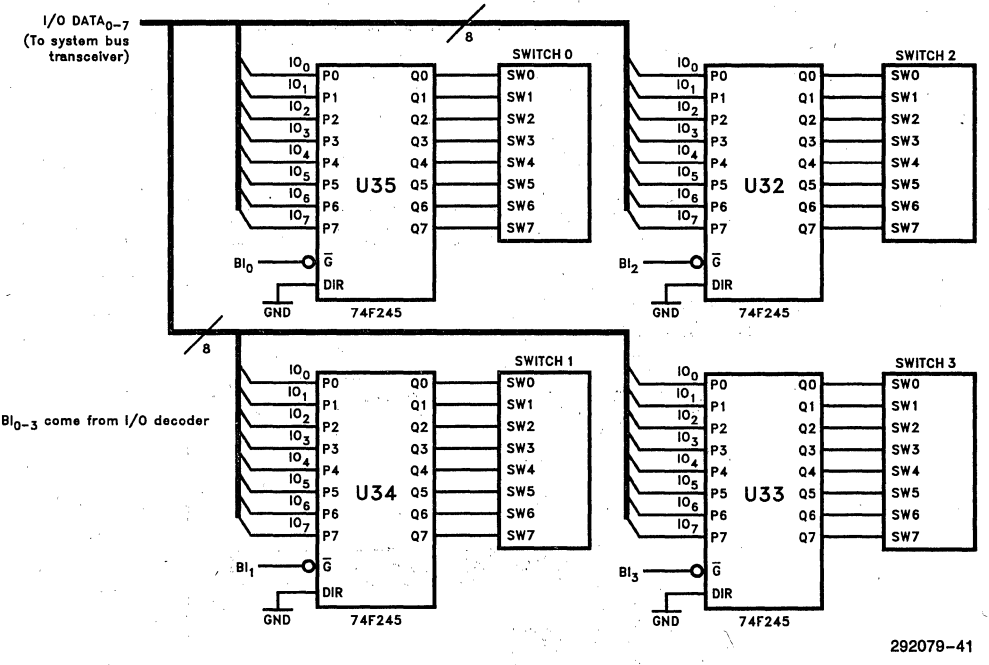
1. To locate the board within the system I/O space, and
2. To identify the board version to assure the software matches the hardware.

The hardware consists of 4 DIP switches and associated 74F245 transceivers (Figure 24, U33–U36). Each switch is read by selection of its I/O address (Figure 15, use $\overline{BI_0}$–$\overline{BI_3}$). The DIP switches can be replaced by EPLDs that permanently "hardwire" the settings. In this case, the identifier is changed by reprogramming the EPLDs.

Zero-Wait-State Selection

The zero-WAIT selection feature is only applicable in a PC AT system. Driving a low input to the 0WS pin of the PC I/O channel within 21.5 ns of $\overline{MEMR}$ or $\overline{MEMW}$ going low keeps the system from inserting the standard WAIT-states into the I/O channel bus cycle. On the page memory board, the 0WS signal is generated by the Boolean equation:

$$(\overline{SMEMW} \cdot \overline{SMEMR}) + \overline{MEMDECODE} = 0WS.$$



NOTE:
DIP switches can be replaced with EPLDs.

Figure 24. Hardware Used to Locate and Identify Page Memory Board

Initializing Software for the Paged Memory Board

(The assembly language software is included in the Appendix.)

In the following sections, algorithms will be shown that verify the page-memory board's functionality. To access this board, first find the location of the base I/O address. From Figure 15, the board's I/O ports are accessed as offsets of the I/O base address:

Board Identifier *n*	@ Base Address + *n* (n = 0, 1, 2, 3)
Window Base Address	@ Base Address + 4
Presence Detect Pins	@ Base Address + 5
Master/Slave and $V_{\overline{PPEN}}$	@ Base Address + 6
Page Number	@ Base Address + 7

Next, the Window Base Address I/O port is used to locate the "page" in DOS's memory space. It is then necessary to determine the density of the SIMMs and the total memory available.

Locating the Base I/O Address

Use the board identifiers to locate the base I/O address. The software reads I/O locations until the correct byte sequence is found (Figure 25). Some discretion should be made when choosing the board's I/O address. (See table of I/O port usage in Appendix.) The PC XT and PC AT specification allocates 32 I/O ports at 0300h to 031Fh for prototype cards. We will use this address range for this example. Because the I/O ports for the paged-memory board must begin on an 8-byte boundary, the only possible base addresses are 300h, 308h, 310h, 318h.

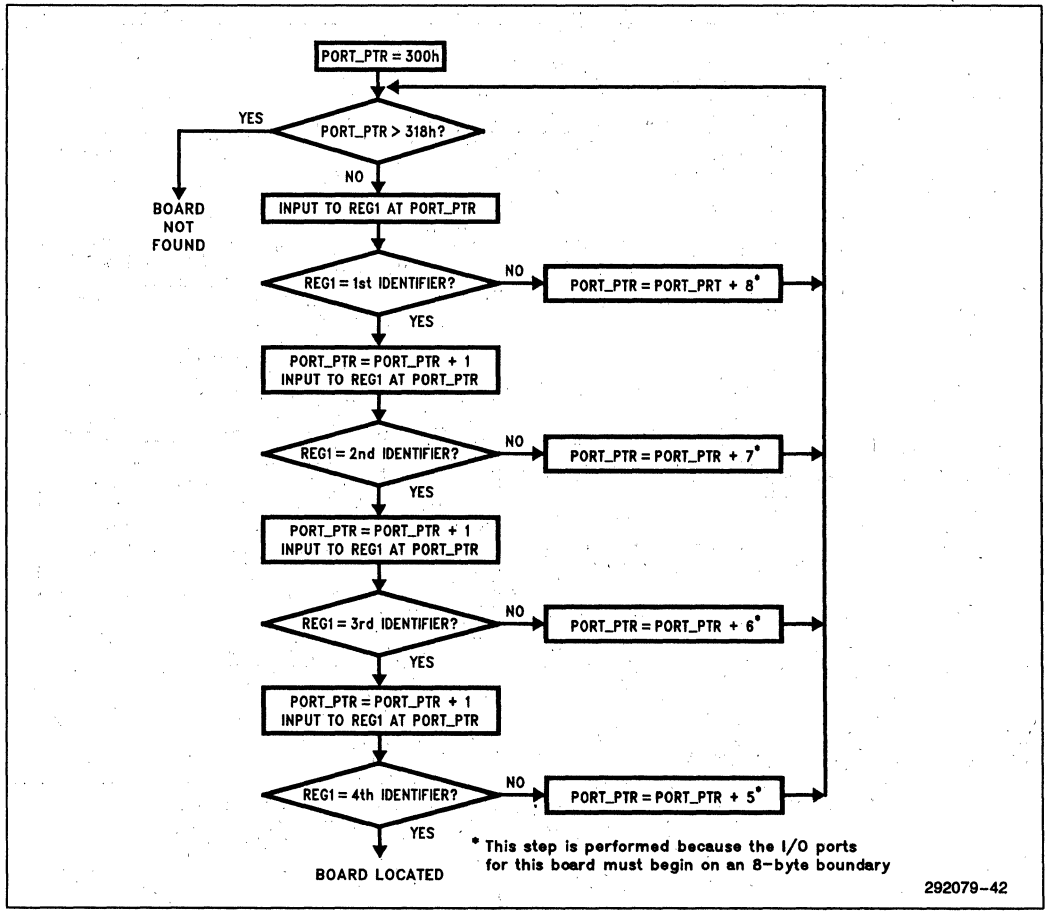


Figure 25. Locating the Page-Memory Board

Locating the Base Memory Address

The base memory address gives the location of the page within the system's memory space. The address switch settings for A_{16}–A_{19} are read from the correct I/O port, Base Address + 4 (Figure 15). After reading these address lines they are stored in the ES segment register used as a pointer to access that memory segment. A_{16}–A_{19} must be shifted into the upper nibble of the ES register to allow proper address generation.

Determining Memory Capacity

First ensure the board is set to read from Page 0. The PD pins are read and translated, using a lookup table of SIMM densities, to a functional value. Then the device identifiers should be read to determine:

1. The number of components on each SIMM;
2. the number of SIMMs installed on the board;
3. and which sockets are used.

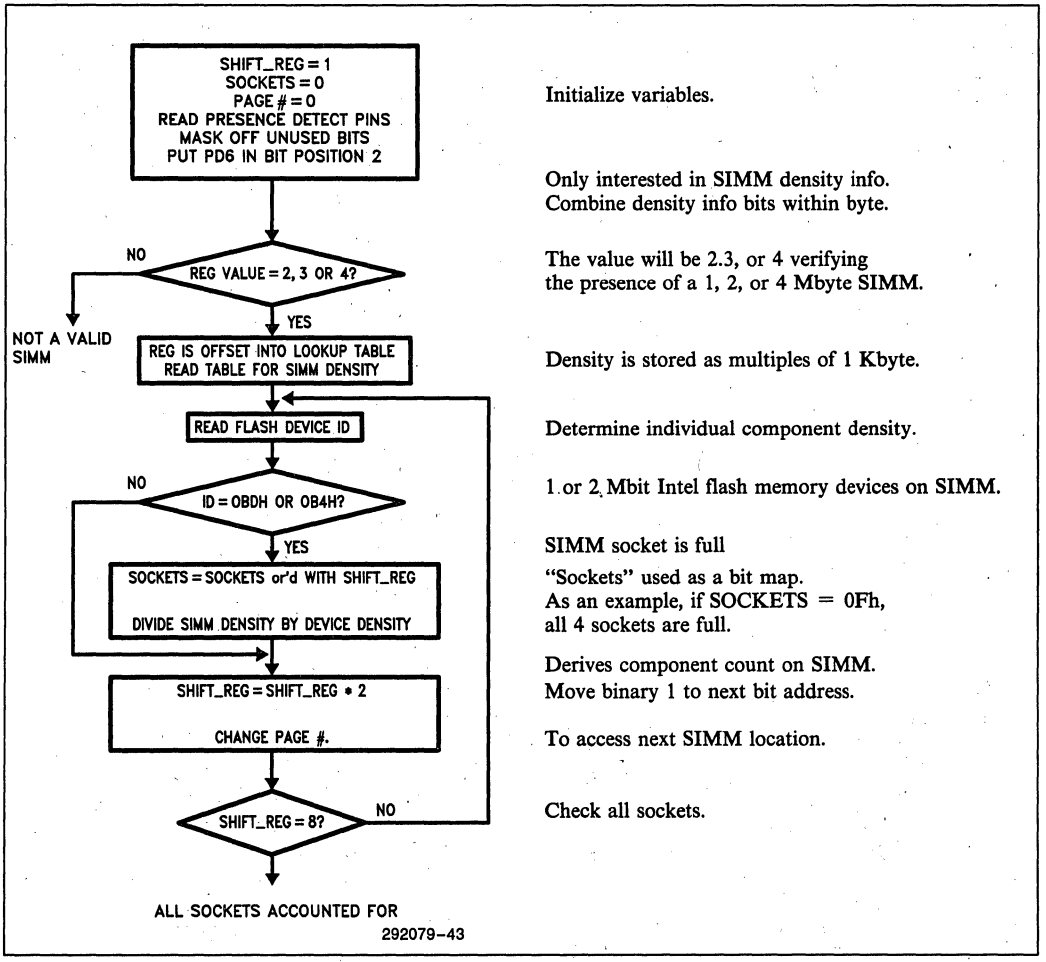


Figure 26. Determining SIMM and Component Densities and Locations

Linear Addressing

Linear addressing directly maps the flash memory array into the system's memory space. "Instantaneous Access" of the entire array is the obvious advantage over paging. Additionally, the decode circuitry is simplified. Figure 27 shows an example for accessing 16 Intel Flash Memory 28F020s arranged in a 4 Mbyte linear array.

The number of address lines used, as well as the decoder type (2 to 4, 3 to 8, etc.), is determined by the flash memory device size. The address lines A_1–A_{18} are used for byte selection within each device (256 Kbytes * 8).

The decodes for the individual devices can be designed in a row-column method similar to that used for the page memory board. An alternative design uses an individual chip enable for each of the 16 devices.

The enable for the 74HC138 (3 to 8 decoder) is governed by a 74F521 comparator. System address inputs to the comparator are chosen to locate this array on a 4 Mbyte boundary. (The array base address could be located on a non-4 Mbyte boundary but this would add to the decoding complexity.) With the inputs chosen in this example (A_{22}–A_{23}), the array base address will be between address 0 and 12 Mbytes to confine this memory array within the PC AT defined address space of 16 Mbytes. A_{19}–A_{21} are inputs to the decoder which generates one of the eight chip enables ($\overline{CE}$). (Use a 74F245 transceiver for the data bus of every 8 flash memory devices. The address lines also need buffering when connected to a PC bus.)

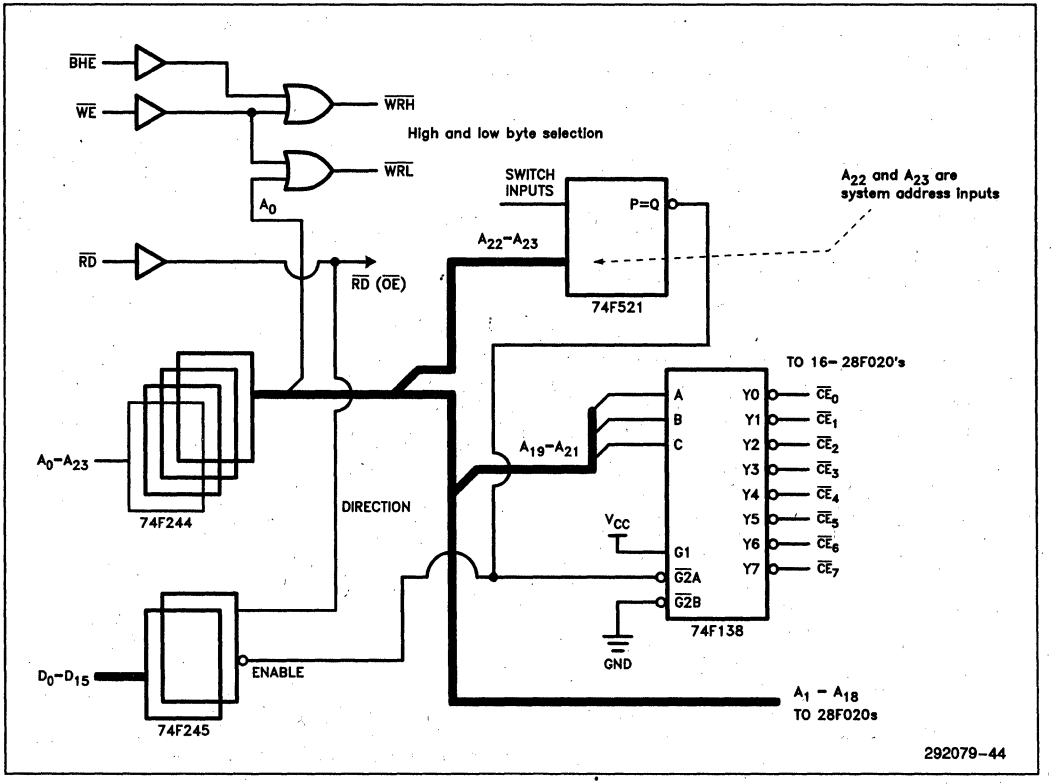


Figure 27. Linear Addressing Hardware Block Diagram

I/O Addressing

From the standpoint of the system's address space usage, I/O addressing provides a conservative solution. As an example, four gigabytes of a flash memory array can be addressed through only two I/O ports. An I/O write sends the flash memory addresses out on the data bus. This "data" is latched (using '574s) and made available to the flash memory devices and decoding circuitry (Figure 28). A third I/O port, used as an enable for the flash memory device decoder and transceivers, helps conserve power when the array is not being accessed.

Relative to linear addressing, I/O addressing generally has limited access speed capability because of the I/O "bottleneck". Read speed can be increased to match linear addressing by replacing the '574 latches with '191 counters.

In the following circuit example, decoding for I/O is accomplshed with a 74F138, 3 to 8 decoder (Figure 29, U1). The base address for these I/O ports is on an 8-byte boundary. When any one of the 8 I/O addresses is selected, the comparator (U2) generates the enable signal (if AEN is low) for the decoder.

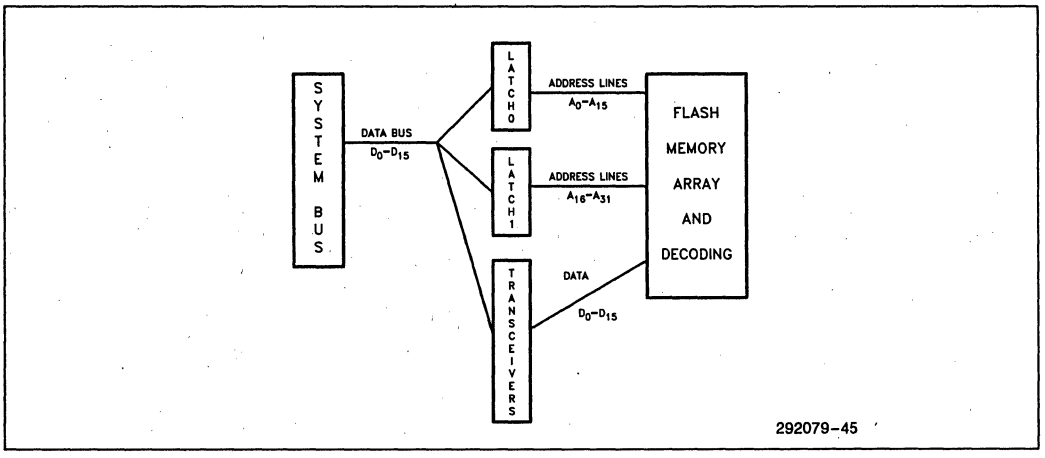


Figure 28. Data Bus Generates Flash Memory Addresses

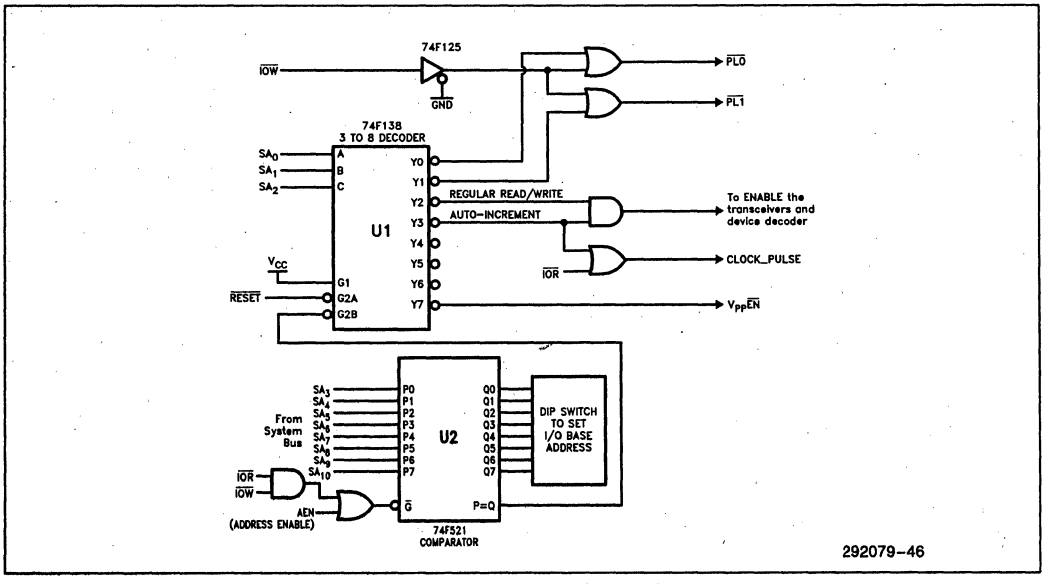
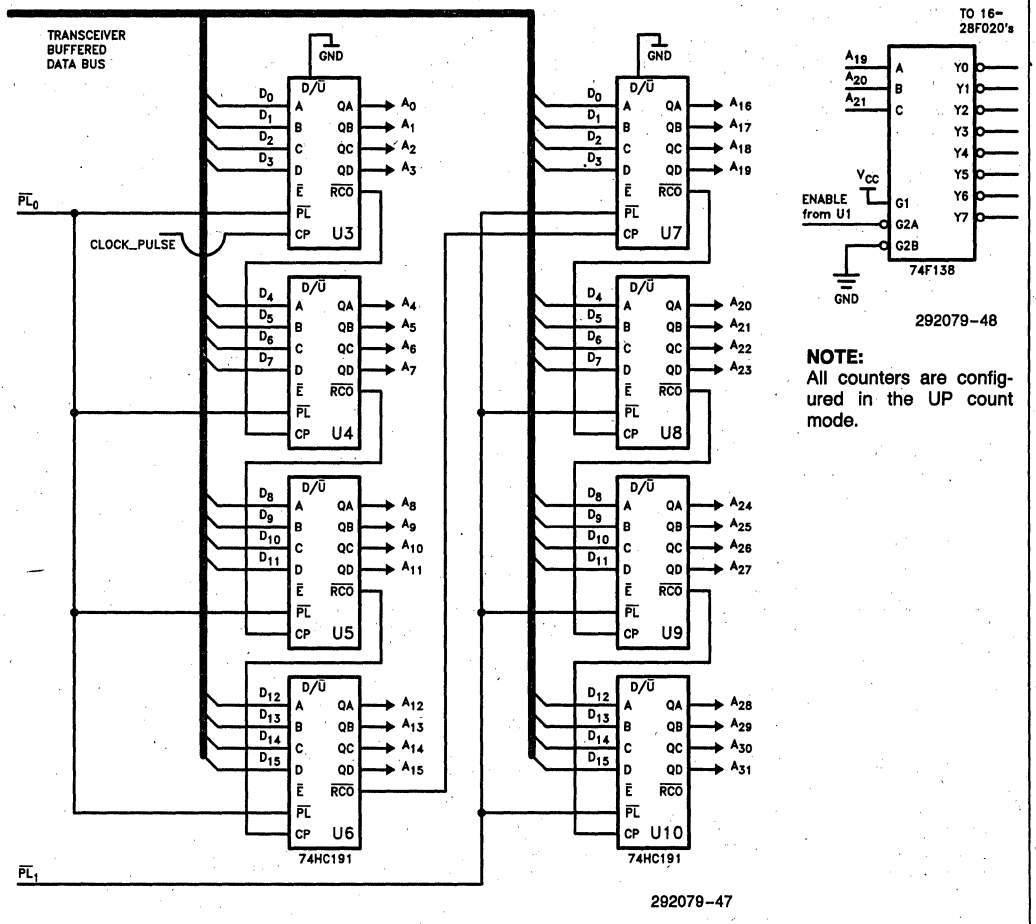


Figure 29. I/O Decode and Enable Circuitry

An I/O write to the first and second ports generates parallel load signals, $\overline{PL_0}$ and PL_1. These signals latch the "data" (addresses) into the 4-bit counters (Figure 30, U3–U10). This latched data represents the address for the flash memory devices.

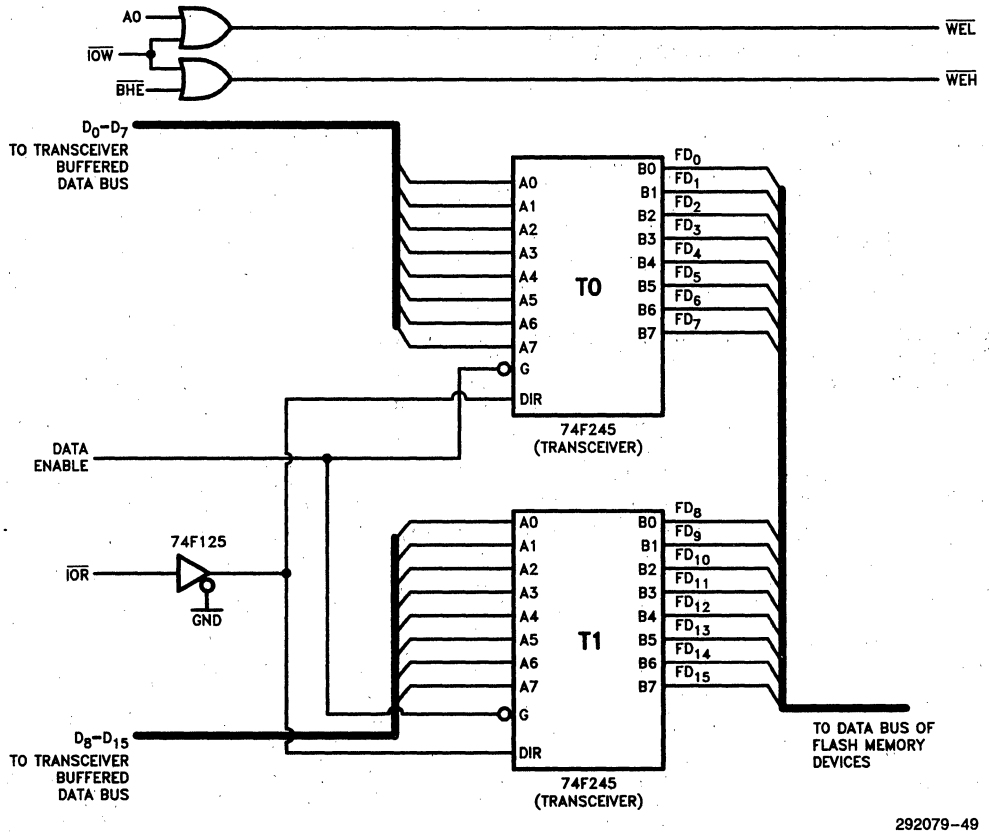
A read or write from the selected flash memory address is performed when the third I/O port is accessed (Figure 29, U1); this generates an enable for the flash memory device decoder and associated transceivers (Figure 31, T_0 and T_1).



NOTE:
All counters are configured in the UP count mode.

NOTE:
A_0–A_{31} are inputs to flash memory devices. Only address lines A_0–A_{18} are used for the 28F020s.

Figure 30. Counter Circuitry



NOTE:
The 16-28F020's are arranged as a 16-bit word configuration. $\overline{WEL}$ and $\overline{WEH}$ are for the low and high bytes, respectively.

Figure 31. Transceiver Enable Circuitry

The fourth I/O port activates the circuitry that obtains very high performance from an I/O board. A read from the fourth I/O port address generates the clock signal for the 74HC191s, CLOCK__PULSE. The counter increments on the rising edge of the clock (read signal), selecting the next flash memory address. This rising edge occurs at the end of the I/O read cycle and the data has already been read. This method is analogous to address pipelining. It is perfect for a "string" read because continuous reads from the fourth I/O port automatically increments the address to access the next word of data stored in the flash memory array.

Capacitive Loading

Capacitive loading is an important consideration for a solid-state mass storage device. If proper buffering techniques are not followed, performance degradation will occur.

The specifications for Intel's Flash Memory devices are based on a test capacitive load of 100 pF. Each data line contributes 12 pF, therefore 8 devices connected to one data transceiver will not experience speed derating (12 pF * 8 = 96 pF). Additional flash memory devices

on that transceiver will increase the loading seen by any one device.

Degradation is calculated as follows (Q = Amount of Charge, T = Time, C = Capacitance, V = Voltage, and I = Current):

COULOMBS LAW STATES:

$$Q = I\Delta T$$

AND GIVEN THE RELATION:

$$V = \Delta Q/C \rightarrow I = C\,\Delta V/\Delta T$$

FROM THIS RELATION, THE CHANGE IN ACCESS TIME CAN BE EXPRESSED IN TERMS OF CAPACITIVE LOAD:

$$\Delta T = C\,\Delta V/I$$

For example, using four SIMMs, each with 8 components in a 16-bit configuration (4 components on high byte and 4 components on low byte), each Intel Flash Memory device sees a load of 15 devices (12 pF * 15 = 180 pF). This loading is 80 pF in excess of the device specification so therefore:

$$\text{Time Change} = \text{Additional Capacitance} \times \frac{(V_{CC} - V_{OL})}{I_{OL}}$$

$$= 80\text{ pF} \times \frac{(5.0 - 0.4)\text{V}}{5.8\text{ mA}} = 64\text{ ns}$$

(Reflecting worst case conditions.)

SOFTWARE DESIGN IMPLEMENTATIONS

Each hardware implementation discussed above can be used in several types of mass storage applications. The general categories include: data recoders, Write-Once-Read-Many (WORM) drives for storing application programs and fixed data, and magnetic disk emulators.

Data Recording

The applications for data recording represent an endless list. Examples include digital imaging, digital photography, point-of-sale terminals, patient monitors, and flight recorders. These systems will use Intel Flash Memory as a more economical and reliable replacement for SRAM + battery. Alternatively, mechanical disks will also be replaced by Intel's Flash Memory when higher reliability, lower power consumption, higher performance, and lighter weight are required.

Interleaving

Although the basic concept of data recording is similar from system to system, variations in implementation exist. For instance, some applications require high-speed data acquisition. Data programming rates are improved considerably by employing interleaving techniques. The majority of time spent programming or erasing a flash memory device results from the delay times in the software algorithms. (It is advised to review the standard algorithms first. See any Intel Flash Memory data sheet for Quick-Pulse Programming™ algorithm.) Interleaving takes advantage of these delay times to begin programming consecutive devices.

There are hardware and software mechanisms for interleaving. The flash memory array for hardware interleaving requires special decoding techniques (Figure 32). Contrary to linear decoding, the system address lines A_0–A_3 are decoded to provide the chip select signals and individual bytes are selected with the address lines A_4–A_{20}. (For the Intel 28F010.) This decoding technique allows software to automatically access sequential devices by writing or reading sequential memory addresses. (Data accumulated with program interleaving will not be stored consecutively within a single device.)

The interleaving algorithm to program the 2 Mbyte flash memory array is shown in Figure 34 and 35. The basic goal is to utilize the delay times. To simplfy the algorithm for this discussion, the data will be programmed on a byte-wide basis. Word-wide and double word-wide techniques, discussed later, will further increase programming speeds.

During multi-component programming, the number of pulses required could vary between different devices. Code is reduced if the programming loop does not have to selectively "decide" if a byte has programmed correctly (verified). However, continual programming of a programmed byte is not necessary and should be avoided. This is done by masking the command sent to that particular device. The RAM table in Figure 33 is used as a data and flash memory command buffer. After a programmed byte has verified, its associated data and commands in the RAM table are written with the value 0FFH (RESET command for Intel flash memory). The data is also written as an 0FFH since this is null program data.

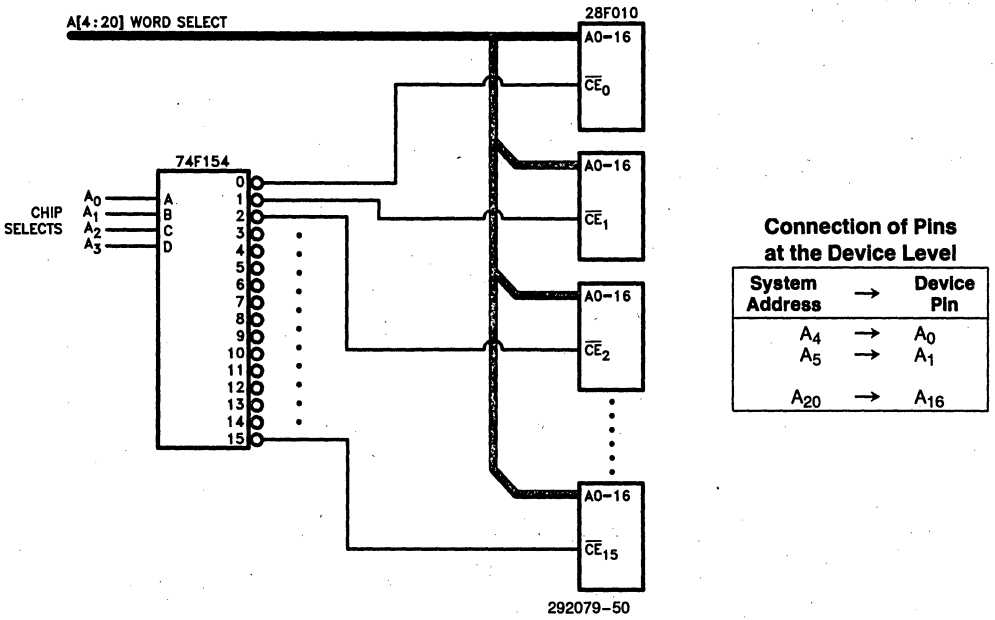
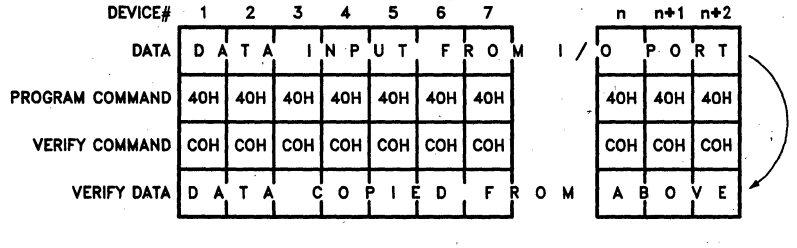


Figure 32. Hardware Interleaving Block Diagram



NOTE:
n = 14 for the example shown in text.

Figure 33. RAM Array Used as Data Buffer and Command Mask Storage

4

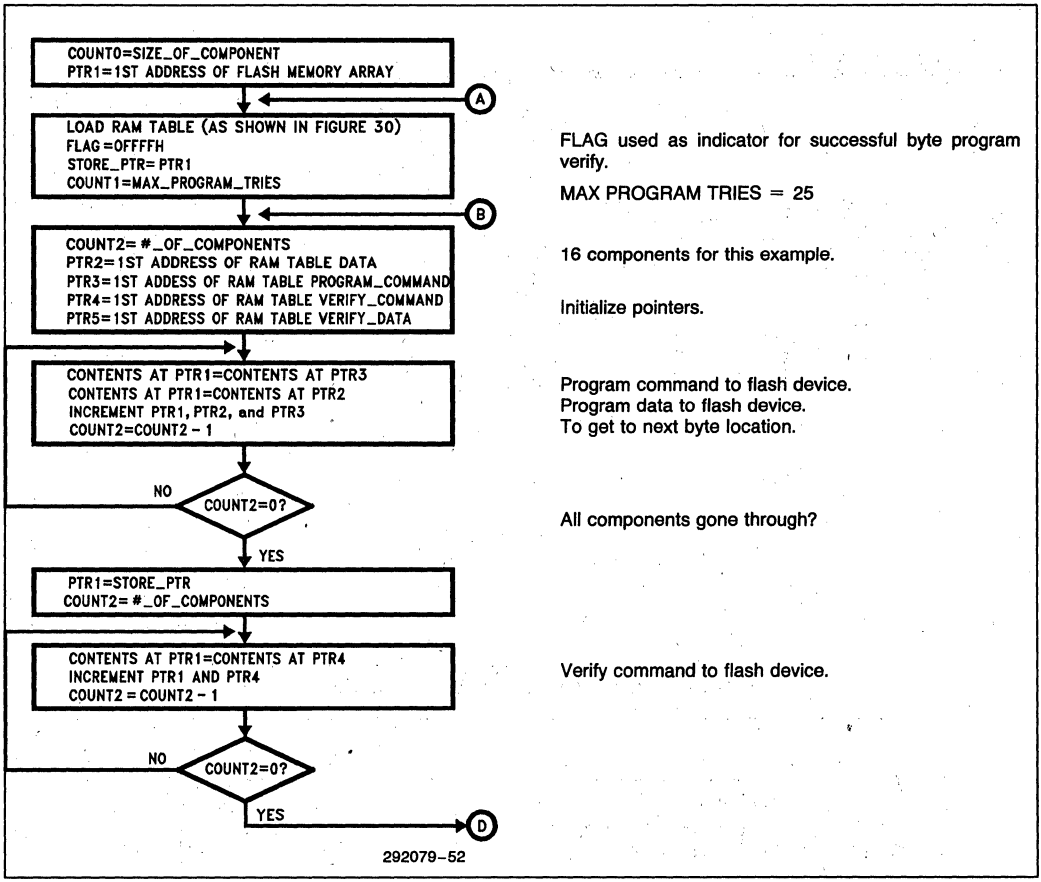
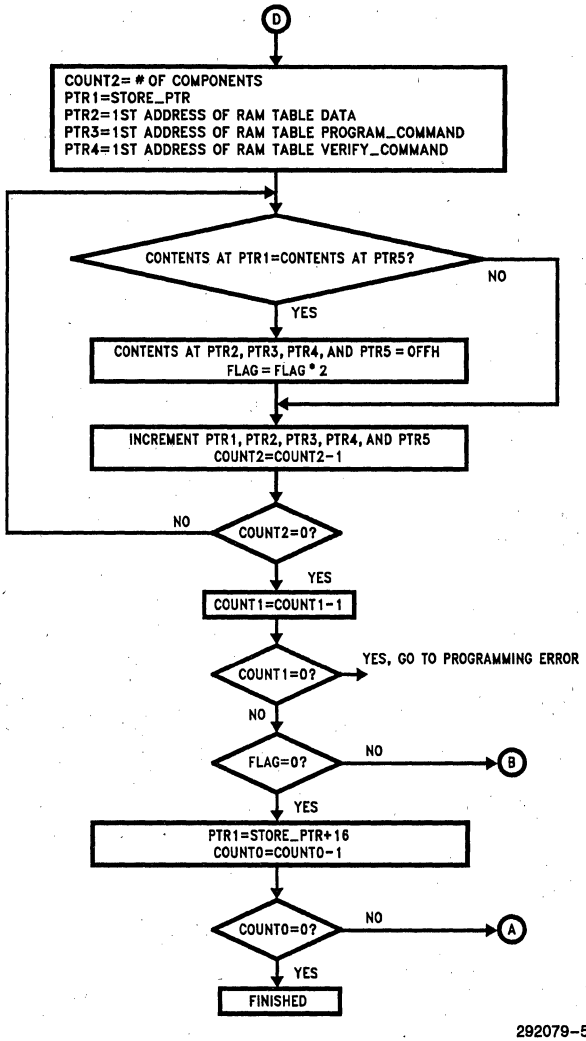


Figure 34. Program Interleaving Algorithm



Does Flash Memory Byte = Verify Byte?

Shift bits of flag left.

Exceeded program tries, go to error.

All 1's shifted out?

To access next byte of each device.

292079-53

Figure 35. Program Interleaving Algorithm (Continued)

Software and hardware interleaving are very similar. Software interleaving is performed using conventional decoding and addressing methods. Instead of incrementing flash memory addresses by one to access the next byte (as with hardware decoding), the address is incremented by the size of the component. While allowing the use of "general-purpose" (non-interleaved) hardware, software interleaving requires reading back the data in the same, non-sequential fashion as was used for recording.

Interleaved erase is useful for erasing an array of flash memory devices. This approach greatly reduces the total subsystem format time. As specified in the erase algorithm, each erase pulse requires a 10 ms delay. (See Quick-Erase™ algorithm in Intel Flash Memory data sheet.) Without interleaving, the processor is idle during this delay time. As with program interleaving, this time is used to begin the erasure of consecutive devices, thereby reducing the overall erase time.

Further program and erase time can be saved by supplementing the byte-wide algorithm with 16- or 32-bit interleaving. Extra data and commands are added to the RAM Mask Table. The major difference in the algorithms involves the verify operation. Depending on the bus width, 2 or 4 bytes are verified simultaneously as shown in Figure 36 (for a 16-bit algorithm).

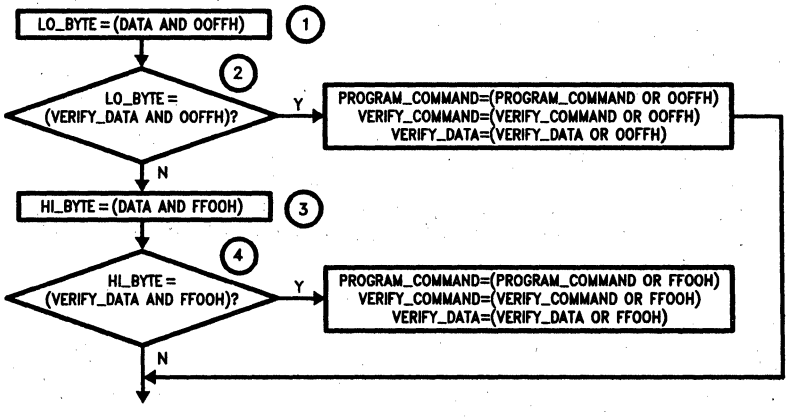
Power Requirements for Interleaving

Current consumption is an important consideration for interleaving. During programming, each device typically consumes 9 mA (1 mA I_{CC} and 8 mA I_{PP}) while programming or erasing; this translates to about 100 mW. If interleaving with 16 devices, about 144 mA (16 devices * 9 mA) or 1.6W, is drawn. Battery powered systems will have a practical limit on the number of components in the interleaving loop. Failure to accommodate these current levels, resulting in V_{PP} voltage drop, will compromise programming and erase reliability.

Write-Once-Read-Many (WORM) Drives

The optical disk is an example of a typical WORM drive application. Its strengths are extremely high densities and low cost per bit. However, it is an unacceptable solution for a low powered, lightweight laptop computer system. It is this environment that solid-state drives offer the greatest benefit. Solid-state ROMs have historically been used in laptop systems to store software programs that seldom change. When the software does change, the ROM "application hardfile" is discarded and a new one is programmed.

Unlike the ROM drive, Intel Flash Memories can be reused and reprogrammed in a true WORM fashion. A computer user can load favorite software programs on the flash memory drive. Adding revised programs to the drive is accomplished by writing to the next free space or by erasing and reprogramming the entire drive. Software drivers can be written to implement this functionality in most operating systems.



NOTES:
1. MASK the HI Byte with 00H.
2. If the LO Byte verifies, then mask the data, program, and verify command with 00FFH (RESET).
3. Mask the LO Byte with 00H.
4. If HI Byte verifies, then mask the data, program, and verify command with 00FFH (RESET).

Figure 36. 16-Bit Masking for Verify Operation

Disk Emulation

Microsoft has a flash memory file system for DOS. It stores and retrieves data or application programs in a manner that, to the end user, appears similar to a disk drive. New files are written sequentially from beginning of memory. However, when the disk is full, it reclaims memory space for storing additional files.

When an application accesses a disk through INT 21H, the MS-DOS* kernel checks the drive letter (Figure 37). If the drive has been declared as a flash memory disk, a built-in redirector services the call. (This is very similar for networked drive accesses.) Otherwise, if the drive letter is that of a floppy or hard disk, the call is handled by the standard DOS file system. The File System provides the link between DOS and the Flash Memory and Hardware device driver. It changes DOS file system commands into a form understood by this unique file structure.

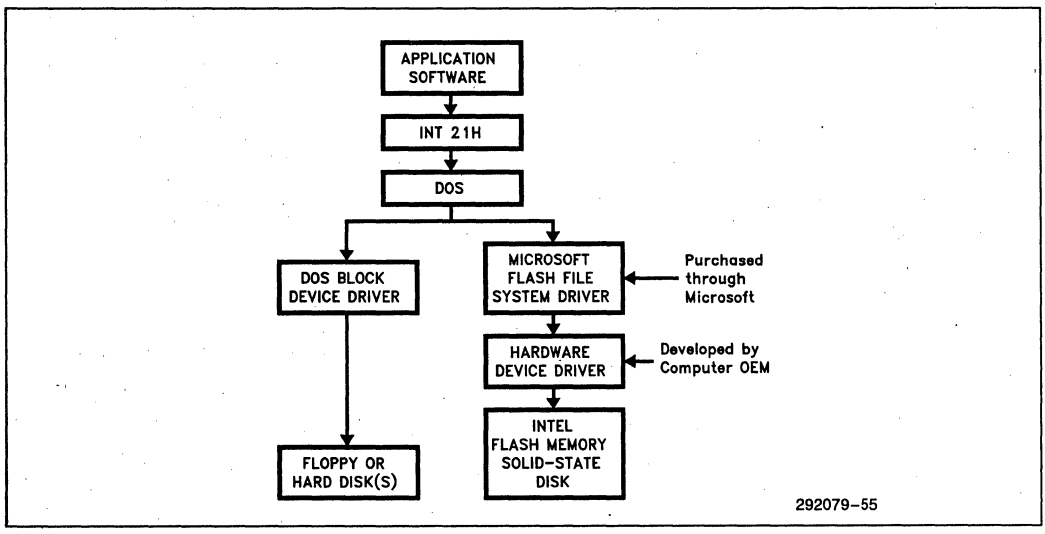


Figure 37. Disk Interface Levels

4

*MS-DOS® and Microsoft® are registered trademarks of Microsoft Corporation.

The Flash File System Driver is the "intelligence" of this file system. It searches for:

1. A Boot Record that identifies the file system and version, and locates the start of the data area;
2. The Root Directory Entry Record and many Directory and File Entry Records.

The file system driver is independent of the hardware interface to the flash memory disk. The hardware device driver, developed by the OEM or BIOS software vendor, interfaces the flash memory disk to the flash file system. It is responsible for the low level calls to the Intel flash memory devices. The actual implementation of the interface is dependent on the hardware configuration of the disk (I/O, paged, and linear addressing are examples).

To minimize fragmentation losses and allow arbitrary extension of files, the flash memory file system uses variable sized blocks rather than the standard sector/cluster method of more traditional file systems. The fundamental structure employed to offer this flexibility is based on linked list concepts; files are chained together using address pointers located within directory entries for each file.

Files and directories are written to the flash memory disk using sequentially free memory locations—a stack-like operation (Figure 38). Furthermore, file sizes can be variable, abandoning the traditional sector/cluster approach of DOS. When "the stack" is full, (containing deleted files), the intelligent software algorithm performs a cleanup operation to reclaim the "dirty" space.

File and subdirectory information is attached to the beginning of each file, unlike the standard DOS approach of directory and FAT placement. As directory and file entries are added, they are located by building a linked-list. Besides containing the customary fields (e.g., name, extension, time, date of creation, etc.), a directory and file entry contains a status byte and various pointers used for the linked-list process. The status byte, besides indicating whether a file/subdirectory exists or is deleted, is also used to signify valid sibling and/or child pointers and to determine if a directory entry pertains to a file or a directory.

When a directory or file is requested or added, the flash memory disk is searched beginning at the head of the linked-list. The chain is followed from pointer to pointer until the correct entry is found. If the search arrives at the chain's end (an FNULL is encountered), the system responds analogously to DOS with a "File not found" message.

This linked-list chain consists of two basic types of pointers: sibling and child. Sibling pointers are used to locate directories or files at the same hierarchial level. Child pointers are used to locate subdirectories or the first file of a particular directory. The following examples elaborate these concepts.

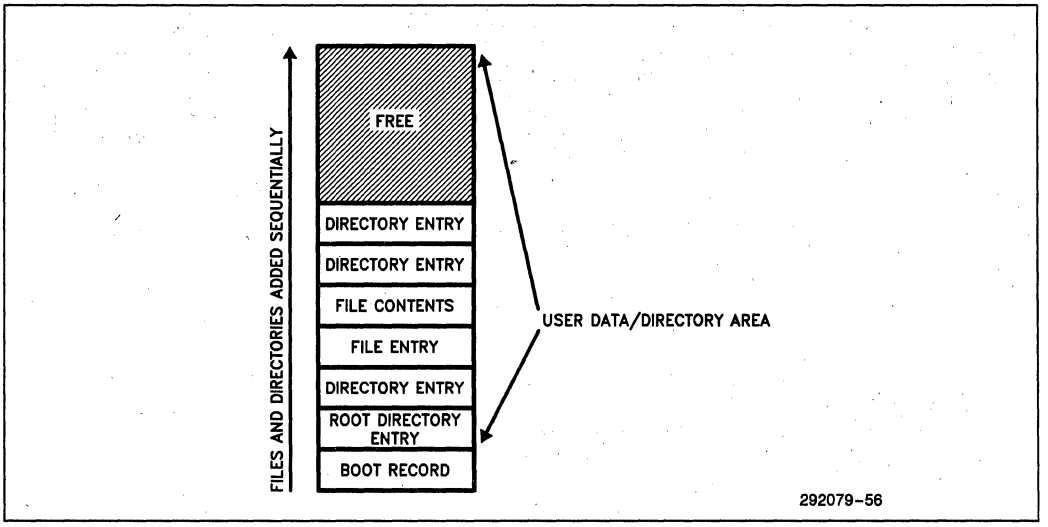


Figure 38. FFS Storage

In Figure 39, Directories B and C are subdirectories of Directory A. Specifically, Directory C is a sibling of Directory B and both are children of Directory A. FNULL indicates the end of the chain.

Figure 40 shows two files (File A and File B) added to a directory (Directory A). File A and File B are at the same level, therefore they are siblings. A file's file entry contains an extent location pointer that indicates the start of its data area.

When a file appears multiple times (because of deleted versions) on the flash memory disk, the file system must find the most recent version. The status byte contains bit fields that indicate whether that particular file is a valid or deleted file. The directory information of a deleted file is used for pointers of the linked list and the search would proceed until the most recent version is found.

A key point to be made for using this method of file storage is that the user is in control of the rate in which the disk becomes full; using the flash memory disk predominantly for application code storage and non-temporary data files reduces the frequency of "cleanup". However, flash memory will typically perform 100,000 cycles and eliminates reliability concerns when used as a hard or floppy disk replacement.

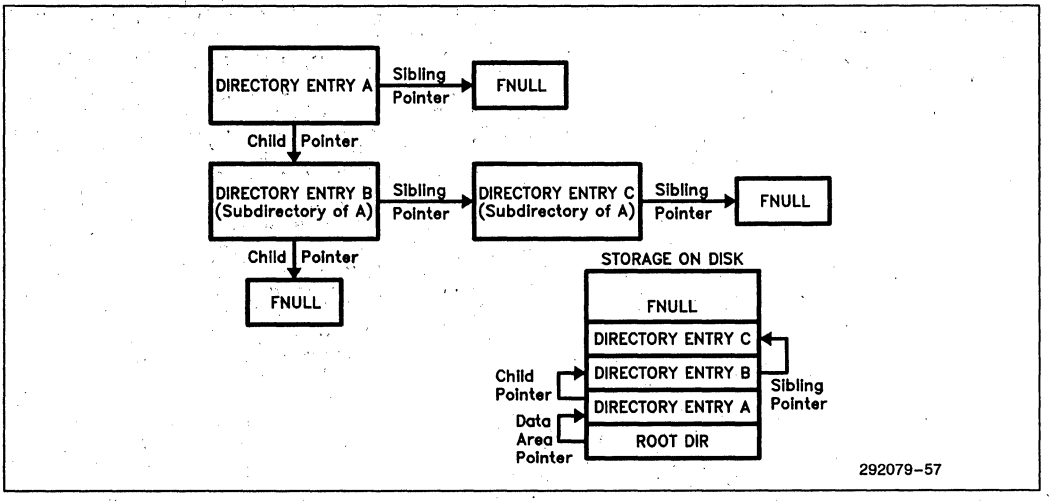


Figure 39. Directory Arrangement

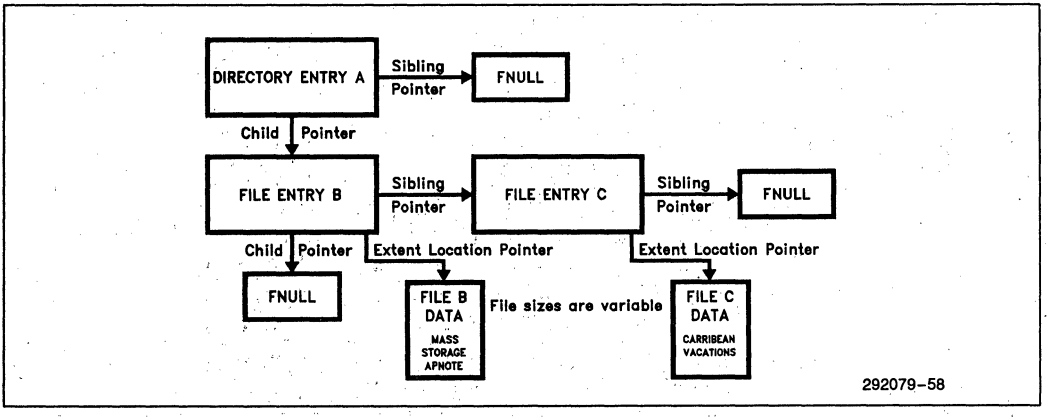


Figure 40. File Arrangement

Creating a Bootable Drive

The startup time of the PC can be decreased by booting from a flash memory disk instead of the magnetic disk. To do this, a "disk-image" is installed on the flash memory disk which is located in the system memory space between C0000H and E0000H (Figure 10). (The "disk image" contains the Boot Record, Directory, and FAT.) This memory space is referred to as Expansion ROM. During the system Power-On-Self-Test (POST), the system searches this memory area for the ROM adapter signature, 055AAh, marking the beginning of the disk image. Once this signature is found, the BOOTSTRAP process begins. The software to create and install this "disk image" is available as a product from Microsoft Corporation as ROM executable MS-DOS.

WHY FLASH?

CHARACTERISTICS OF INTEL FLASH MEMORY

Power consumption, weight, performance, and reliability are the key criteria for a system design. The discussion of Intel flash memory as a mass storage medium would not be complete without a performance analysis and comparison to other technologies.

Power Consumption

Portability of a computer demands battery longevity and consequently minimal power consumption. Small form factor disk drives are being designed specifically for the size and power requirements of laptops.

A drive has three basic operating modes: active, power savings, and standby. The active mode consists of reading, writing, and ready. Ready condition allows "instantaneous" transitions into the read or write states. In the power-savings mode only the drive motor continues to run. Standby shuts off all functionality except for the circuitry needed to "spin-up" the drive. From the standby mode, extra power and considerable time, is required to "spin-up" the disk.

Power Consumption Comparison (Watts)

(Based on typical performance characteristics. The 20 Mbyte Flash Memory disk is based on the use of 80-28F020s. Only two of the forty devices are accessed at a time, the remainder are in standby mode.)

Active Modes	Hard Disk Drive (2.5", 20 Mbytes)	INTEL Flash Memory (20 Mbytes)
Ready	1.7–2.0	0.05 (Same as Standby)
Read	3.5-4.0	0.15
Write	3.5–4.0	0.25
Power Savings	1.5	0.05 (Same as Standby)
Standby	0.1–0.5	0.05
Spinup (from Standby)	9.3	0

For a battery-powered system, 3–4 hours of operation is unacceptable. Battery longevity is achieved by using Intel Flash Memory solid-state storage as a disk replacement. The following table relates battery life and the different functions of disk operation. A "AA" battery with a capacity of 2215 mAH is used for the comparison. Obviously, for a truly accurate representation, other components of the system should be included. But from the data storage point of view, the flash memory disk will operate many more hours than the hard disk drive on a set of batteries.

Hours of Operation for a "AA" Battery (Based on Data from Previous Table and 2215 mAH Battery Capacity)

	Hard Disk Drive (2.5", 20 Mbytes)	INTEL Flash Memory (20 Mbytes)
Read	0.83	22.15
Write	0.83	13.29
Standby	6.64	66.45

Data Access Time

Reading data from a magnetic disk is a very slow process compared to a solid-state disk (SSD). Disk transfer time is lengthy due to four time components: spin-up, seek time, latency, and data transfer time. Spin-up is a factor to consider for battery-powered systems, where most disk accesses are begun from the standby mode. During the seek time, the arm is repositioned to the correct track. Latency is the delay from arm repositioning until the first sector of the transfer moves under the

read/write head. This is dependent on the speed of rotation. The actual transfer of data is the third component. The standard SCSI interface transfers data between 5 Mbits and 10 Mbits per second, with which flash memory compares very favorable.

For this example it is reasonable to assume a transfer rate of 1.0 Mbytes per second. Using a full word-wide (x16) bus bandwidth (120 ns access speed of the device), flash achieves a transfer rate of 16.6 Mbytes per second.

Read Speed Comparisons

	Hard Disk (Standard SCSI Interface)	Floppy Disk	Flash Memory (16-Bit Bus, 120 ns Access)
Seek Time	28 ms		0
Latency	8.3 ms	100 ms	0
Transfer Rate	1.0 Mbyte/s	62 Kbyte/s	16.6 Mbyte/s
Total for 10 Kbyte File	46.54 ms	261.3 ms	0.62 ms

(Floppy disk drive specifications combine access into one category.)

In this example, the flash memory disk has 75 times the read performance over the hard disk. Smaller files result in even greater differences. Additionally, the 5 second spin-up of the hard disk gives the flash memory disk over 8,000 times the performance!

A byte will typically program in Intel Flash Memory in one pulse. (See Intel Flash Memory Data sheet for programming algorithm.) Based on this and the parameters used in the example above, a 10 Kbyte file is written to the flash memory disk in 87.04 ms. Because writes to a hard disk typically begin from spin-down, the flash memory disk is still over 50 times faster. Since reads are 80% of disk access, flash memory's user-perceptible performance advantage is substantial.

Reliability

The definition of hard disk mean-time-before-failure (MTBF) is extremely ambiguous. There are no industry-wide standards for making a reliable calculation. Disk drive manufacturers choose whichever method best suits their product's reliability perception.

One method uses the overall mean failure. The MTBF of all critical components is computer analyzed and the lowest one is selected. A second method tests 100 drives. The hours of the first ones to fail are multiplied by the number of drives. How many reads or writes are performed? Is the disk stopped and started during the process? Standard answers do not exist.

The vagueness of the test procedures makes it difficult to compare the MTBF for a flash memory solid-state disk and a hard disk. Based on the fact that disk usage is 80% reads and 20% writes, a reasonable comparison can be made. (What is not taken into account is that disks are an 'infinite' write, but finite read medium. Continuous reading causes reduced magnetic field strength, a failure mechanism hidden by re-writing the disk.)

Intel's Flash Memory typically performs 100,000 erase/program cycles. (Failure does not occur at this point. The only noticeable change is a gradual increase in program and erase times.) Assume a flash memory disk size of 4 Mbytes that functions like a WORM drive; it is erased and reused after filling.

Based on a typical disk MTBF of 50,000 hours and the 80/20% division, 10,000 hours are used for writing files. Assume the average file size written to disk is 10 Kbytes. A 4 Mbyte flash disk can store approximately 400 × 10 Kbyte files (4 Mbyte/10K = 408) before erasure is necessary.

These 400 files could be writen to the disk 40×10^6 times − (400 files × 100,000 cycles = 40×10^6). The result is that within a 10,000 hour period, one 10 Kbyte file could be written once every 0.9 seconds.

$$\frac{10{,}000 \text{ hours}}{40 \times 10^6 \text{ Files}} \times \frac{3600 \text{ Seconds}}{1 \text{ Hour}} = \frac{0.9 \text{ Seconds}}{\text{File}}$$

It would be more realistic (although still extremely aggressive) to assume that this 10 Kbyte file is written to this disk every 10 minutes. At 100,000 cycles, 40×10^6 files will have been written. The MTBF can be calculated as follows:

$$40 \times 10^6 \text{ Files} \times \frac{10 \text{ Minutes}}{\text{File}} \times \frac{1 \text{ Hour}}{60 \text{ Minutes}} = 6.6 \times 10^6 \text{ Hours}$$

This is an MTBF of over 6 million hours! (See Reliability Report RR60 for more details.)

A flash memory solid-state disk outlasts its mechanical counterpart by at least two orders of magnitude, especially if head parking problems and limited start/stop cycles of the mechanical disk are taken into account.

4

Weight

Lowering the power consumption of your portable system also lowers the weight. Reduced battery demands mean smaller and lighter batteries and power supplies. Weight savings is also gained by the proper choice of the mass storage medium. The small 20 Mbyte 2.5″ disk drives weigh between 9 and 21 ounces. The equivalent capacity of flash memory using 80-2 Mbit TSOPs (which individually weigh 1.16×10^{-2} ounces) weighs 0.93 ounces plus the weight of the circuit board. (See section on Intel flash memory packaging.) This difference is critical when the computer weight requirement is under five (5) pounds.

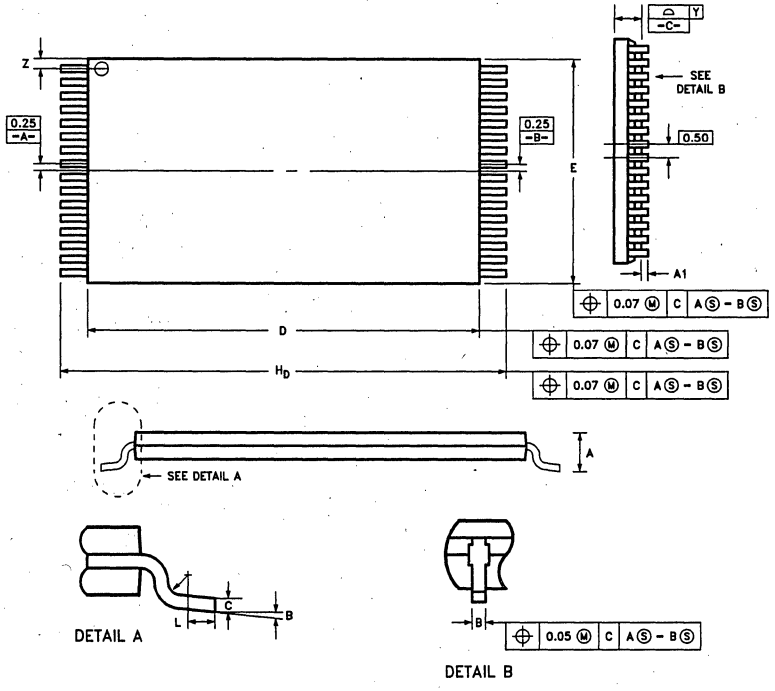
SUMMARY

The advent of Intel Flash Memory has led to the evolution of solid-state mass storage. This application note has provided the building blocks that will allow the innovative manufacturer to remain on the forefront of technology.

- Advanced packaging, such as the TSOP, IC memory cards, and SIMM, is necessary for high-density applications.
- Intel Flash Memory allows flexible system interfacing by using I/O, paged, or linear addressing methods.
- Software variations enable an unlimited number of mass storage applications for Intel Flash Memory.
- Intel Flash Memory offers superior performance over the magnetic disk.

APPENDIX A

28F020 TSOP Dimensions



292079-60

Symbol	Description	Dimensions in mm		
		Min	Nom	Max
A	Package Height			1.20
A_1	Standoff	0.05		
A_2	Package Body Height	0.96	1.01	1.06
B	Lead Width	0.15	0.20	0.30
C	Lead Thickness	0.10	0.15	0.20
D	Package Body Length	18.20	18.40	18.60
E	Package Body Width	7.80	8.00	8.20
H_D	Terminal Dimension	19.80	20.00	20.20
L	Lead Tip Length	0.30	0.33	0.35
N	Lead Count		32	
Y	Seating Plane Coplanarity	0.00		0.10
Z	Lead to Package Offset	0.20	0.25	0.30
Ø	Lead Tip Angle	1	3	5

28F020 TSOP Physical Dimensions Drawings and Specifications

TSOP Sockets and Wands

32-Lead TSOP test sockets are available from the following vendors:

Enplas,
Part Number: 0TS-32-0.5-02
Distributed by:
Tesco International Inc.
1825 S. Grant Street, Suite 745
San Mateo, CA 94402
(415) 572-1683

32-Lead TSOP to DIP adapter sockets are available from the following vendor:

California Integration Coordinators Inc.,
Part Numbers: CIC-32TS-32D-AG-ENP-GANG-S,
CIC-32TS-32D-AG-ENP-GANG-R
656 Main Street
Placerville, CA 95667
(916) 626-6168

Emulation Technologies,
Part Numbers: AS-32-32-01TS-GENP-GANG-R
(F Version-Reverse),
AS-32-32-01TS-GENP-GANG-S
(E Version-Standard)
2344 Walsh Ave., Bldg. F
Santa Clara, CA 95051
(408) 982-0660

Suction wands for transferring units are available from the following vendor:

H-Square Corp.
1289-H Reamwood Avenue
Sunnyvale, CA 94089

1	V_{SS}	21	$\overline{CE3}$	41	A_{11}	61	DQ9
2	V_{CC}	22	$\overline{CE2}$	42	A_{10}	62	DQ8
3	V_{PP}	23	$\overline{CE1}$	43	A_9	63	DQ7
4	$\overline{OE}$	24	$\overline{CE0}$	44	A_8	64	DQ6
5	$\overline{WEH}$	25	V_{SS}	45	A_7	65	DQ5
6	$\overline{WEL}$	26	NC	46	A_6	66	DQ4
7	NC	27	NC	47	A_5	67	DQ3
8	RES	28	NC	48	A_4	68	DQ2
9	RES	29	NC	49	A_3	69	DQ1
10	RES	30	NC	50	A_2	70	DQ0
11	RES	31	NC	51	A_1	71	V_{PP}
12	RES	32	NC	52	A_0	72	V_{CC}
13	RES	33	NC	53	RES	73	PD1
14	RES	34	NC	54	V_{SS}	74	PD2
15	RES	35	NC	55	DQ15	75	PD3
16	RES	36	A_{16}	56	DQ14	76	PD4
17	RES	37	A_{15}	57	DQ13	77	PD5
18	RES	38	A_{14}	58	DQ12	78	PD6
19	RES	39	A_{13}	59	DQ11	79	PD7
20	RES	40	A_{12}	60	DQ10	80	V_{SS}

Figure 43. 1 Mbyte SIMM Pinout

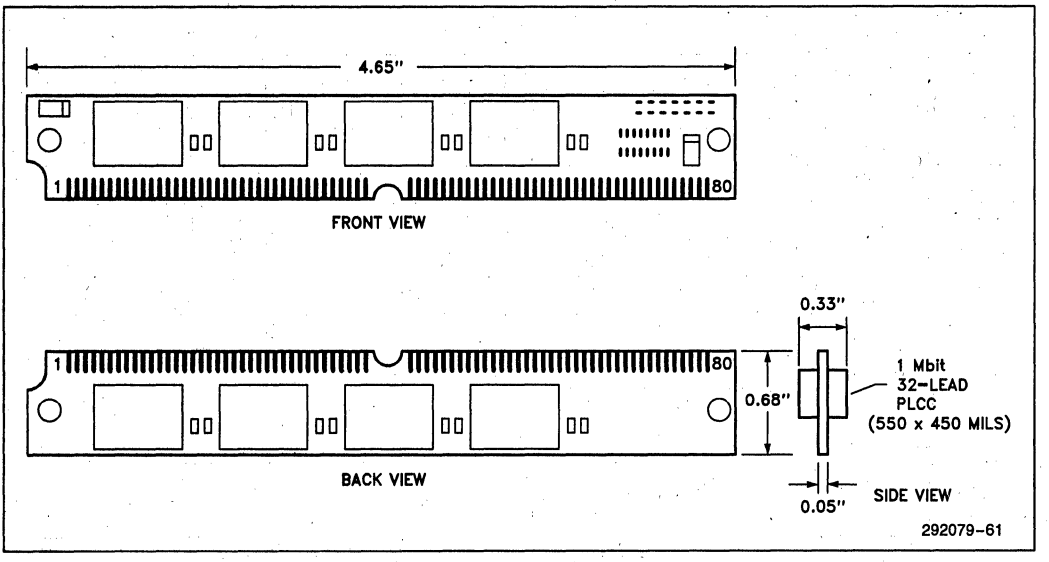


Figure 44. 1 Mbyte SIMM Dimensions

Module Capacity Idenfitication

Module Capacity Word Depth	PD6	PD2	PD1
No Module	O	O	O
256K/32M	O	O	S
512K/64M	O	S	O
1M/128M	O	S	S
2M/256M	S	O	O
4M/512M	S	O	S
8M/1G	S	S	O
16M/2G	S	S	S

NOTE:
These PD pins are JEDEC defined, not future product commitments.

Module Speed Identification

Maximum Access Time	PD7	PD5	PD4	PD3
>300 ns	S	S	S	S
300 ns	S	S	S	O
250 ns	S	S	O	S
200 ns	S	S	O	O
185 ns	S	O	S	S
150 ns	S	O	S	O
135 ns	S	O	O	S
120 ns	S	O	O	O
100 ns	O	S	S	S
85 ns	O	S	S	O
70 ns	O	S	O	S
60 ns	O	S	O	O
50 ns	O	O	S	S
40 ns	O	O	S	O
30 ns	O	O	O	S
ND	O	O	O	O

O = Open Circuit On Module
S = Short Circuit to Ground on Module
ND = Not Defined

```
EPLD ADF for Presence Detect WAIT-State Generator
PLFG Applications    1-800-323-EPLD
Intel Corp.
June 6, 1990
U999
002
85C220
Pre-loadable wait state down counter/READY generator
OPTIONS: TURBO = ON

PART: 85C220

INPUTS: CLK@1, nLOAD@2, PD7@3, PD6@4, PD5@5, PD4@6
OUTPUTS: nREADY@19, Q3@18, Q2@17, Q1@16, Q0@15, nDL@14

NETWORK:
    CLK = INP(CLK)              % System clock input %
    nLOAD = INP(nLOAD)          % Load count input %
    PD7 = INP(PD7)              % PD[7:4] Wait state %
    PD6 = INP(PD6)              % count size inputs %
    PD5 = INP(PD5)              % to lookup table %
    PD4 = INP(PD4)
    nREADY, nREADY = RORF(nREADYd,CLK,GND,GND,VCC) % /READY Output %

    Q3,Q3 = RORF (Q3d,CLK,GND,GND,VCC)          % counter outputs . . . %
    Q2,Q2 = RORF (Q2d,CLK,GND,GND,VCC)          % not externally %
    Q1,Q1 = RORF (Q1d,CLK,GND,GND,VCC)          % necessary %
    Q0,Q0 = RORF (Q0d,CLK,GND,GND,VCC)
    nDL,nDL = RORF (nDLd,CLK,GND,GND,VCC)

EQUATIONS:
    Q0d = Q0EQN * !READY * !COUNT_ZERO          % count if not ready %
        + Q0 * (READY + !LOAD)                  % hold if ready %
        + X0 * LOAD * !READY * COUNT_ZERO;      % read inputs on LOAD %
        Q0EQN = !Q0;

    Q1d = Q1EQN * !READY * !COUNT_ZERO
        + Q1 * (READY + !LOAD)
        + X1 * LOAD * !READY * COUNT_ZERO;
        Q1EQN = Q1 * Q0 + !Q1 * !Q0;

    Q2d = Q2EQN * !READY * !COUNT_ZERO
        + Q2 * (READY + !LOAD)
        + X2 * LOAD * !READY * COUNT_ZERO;
        Q2EQN = Q2 * (Q1 + Q0) + !Q2 * !Q1 * !Q0;

    Q3d = Q3EQN * !READY * !COUNT_ZERO
        + Q3 * (READY + !LOAD)
        + X3 * LOAD * !READY * COUNT_ZERO;
        Q3EQN = Q3 * (Q2 + Q1 + Q0) + !Q3 * !Q2 * !Q1 * !Q0

    nREADYd' = !Q3 * !Q2 * !Q1 * LOAD * !nDL;   % Anticipate counter %
                                                % to provide READY%
    nDLd = nLOAD;                               % hold until LOAD is%
                                                % taken away %
```

EPLD ADF for Presence Detect WAIT-State Generator

```
    COUNT_ZERO = !Q3 * !Q2 * !Q1 * !Q0;
    LOAD = NLOAD';
    READY = nREADY';

    X3 = GND;                                    % Wait State Scrambler %
    X2 = 3CNT;                                   % lookup table %
    X1 = 8CNT + 7CNT + 6CNT + 5CNT;
    X0 = 5CNT;

    8CNT =  PD7 * !PD6 * !PD5 * !PD4;
    7CNT = !PD7 *  PD6 *  PD5 *  PD4;
    6CNT = !PD7 *  PD6 *  PD5 * !PD4;
    5CNT = !PD7 *  PD6 * !PD5 *  PD4;
    4CNT = !PD7 *  PD6 * !PD5 * !PD4;
    3CNT = !PD7 * !PD6 *  PD5 *  PD4;
    2CNT = !PD7 * !PD6 *  PD5 * !PD4;

END$
```

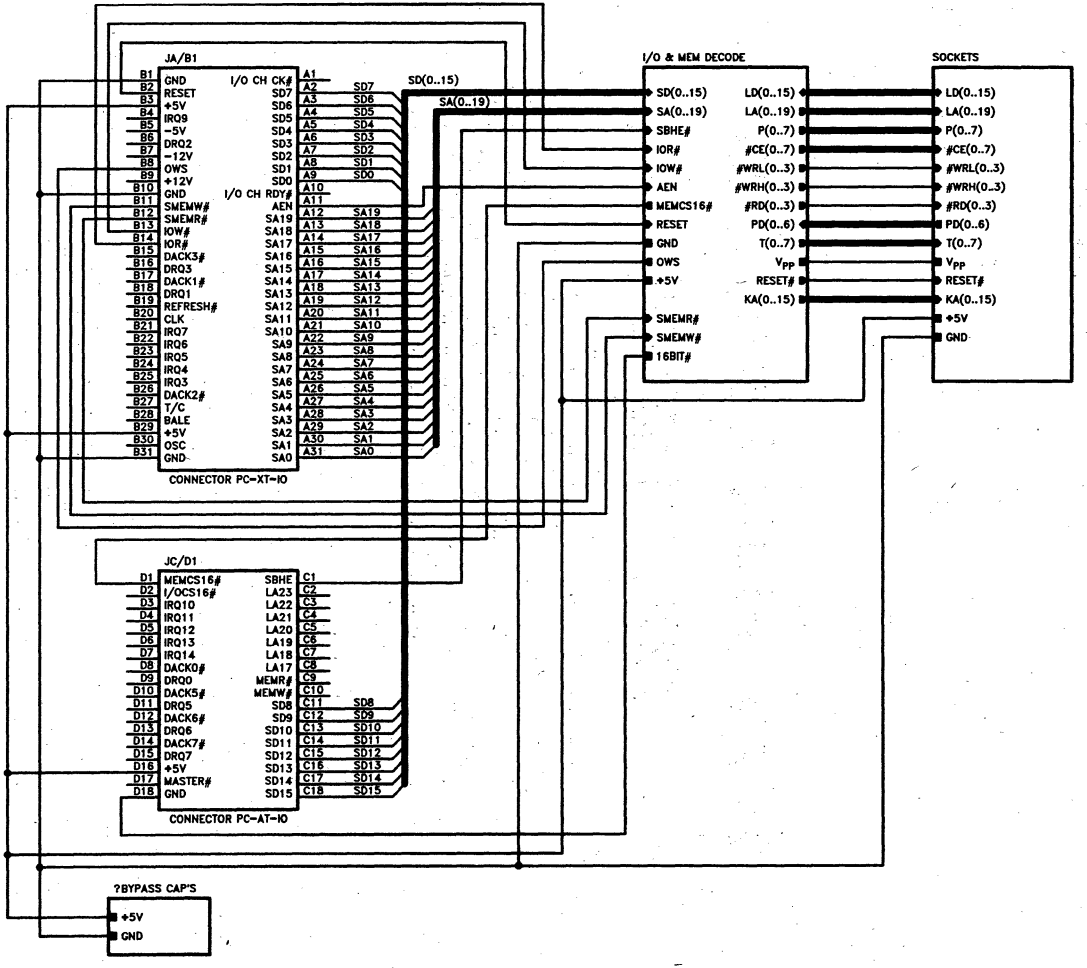
EPLD ADF for Presence Detect WAIT-State Generator (Continued)

Decoding Truth Table for "Multiplexing" Data Bus of PCMCIA/JEIDA Memory Card

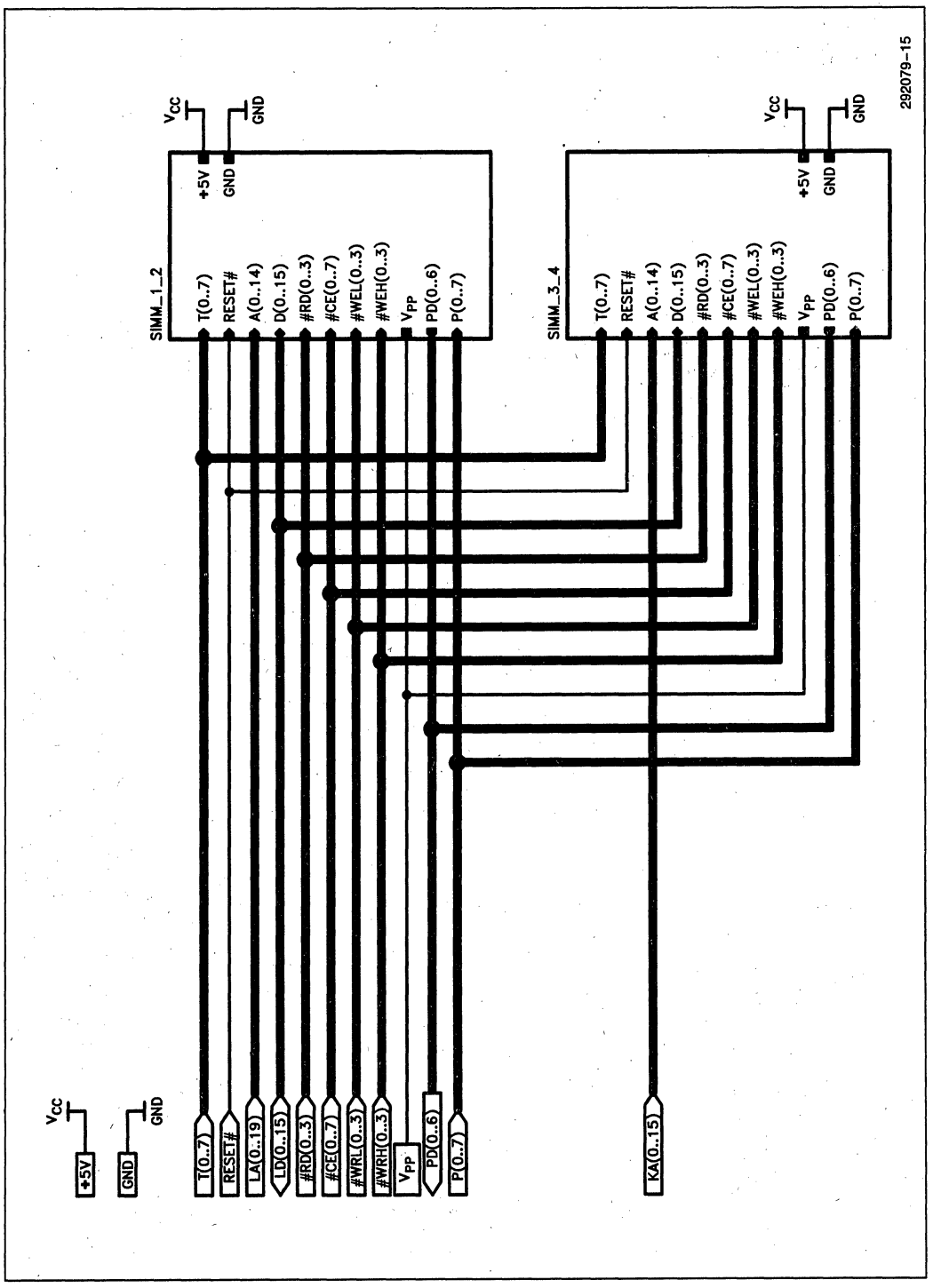
System Bus Width	Data Transfer	$\overline{CSH}$	$\overline{CSL}$	A_0	1	2	3
8 or 16	None	1	1	x	1	1	1
8 or 16	Lo-Byte	1	0	0	0	1	1
8	Hi-Byte to Lo-Byte	1	0	1	1	0	1
16	Hi-Byte	0	1	x	1	1	0
16	Low and High Byte	0	0	x	0	1	0

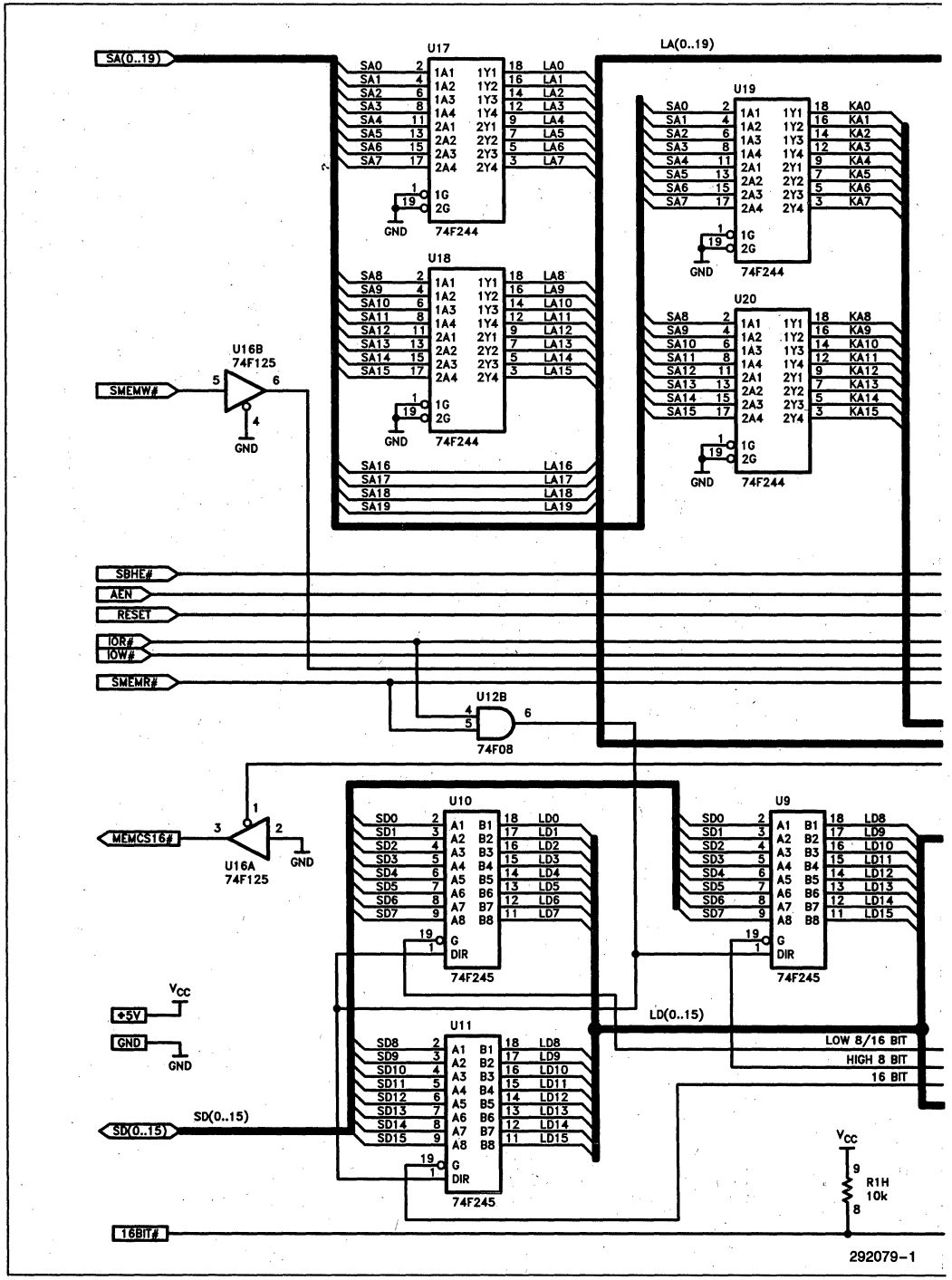
NOTE:
References Figure 8 in Memory Card Section.

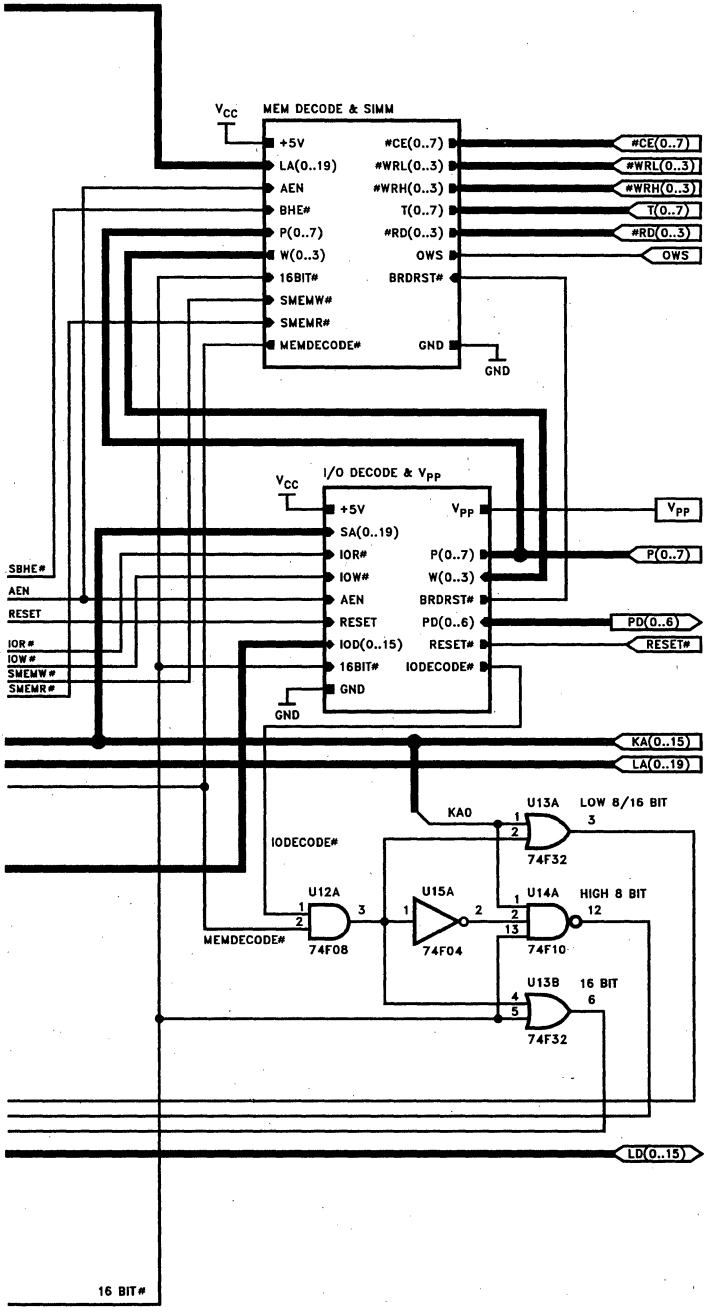
PAGE MEMORY BOARD SCHEMATICS


JA/B1
B1 GND
B2 RESET
B3 +5V
B4 IRQ9
B5 -5V
B6 DRQ2
B7 -12V
B8 OWS
B9 +12V
B10 GND
B11 SMEMW#
B12 SMEMR#
B13 IOW#
B14 IOR#
B15 DACK3#
B16 DRQ3
B17 DACK1#
B18 DRQ1
B19 REFRESH#
B20 CLK
B21 IRQ7
B22 IRQ6
B23 IRQ5
B24 IRQ4
B25 IRQ3
B26 DACK2#
B27 T/C
B28 BALE
B29 +5V
B30 OSC
B31 GND
I/O CH CK#
I/O CH RDY#
AEN
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 A16 A17 A18 A19
SD7 SD6 SD5 SD4 SD3 SD2 SD1 SD0
SA19 SA18 SA17 SA16 SA15 SA14 SA13 SA12 SA11 SA10 SA9 SA8 SA7 SA6 SA5 SA4 SA3 SA2 SA1 SA0
CONNECTOR PC-XT-IO
SD(0..15)
SA(0..19)
JC/D1
D1 MEMCS16#
D2 I/OCS16#
D3 IRQ10
D4 IRQ11
D5 IRQ12
D6 IRQ13
D7 IRQ14
D8 DACK0#
D9 DRQ0
D10 DACK5#
D11 DRQ5
D12 DACK6#
D13 DRQ6
D14 DACK7#
D15 DRQ7
D16 +5V
D17 MASTER#
D18 GND
SBHE
LA23 LA22 LA21 LA20 LA19 LA18 LA17
MEMR#
MEMW#
SD8 SD9 SD10 SD11 SD12 SD13 SD14 SD15
C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18
CONNECTOR PC-AT-IO
?BYPASS CAP'S
+5V
GND
I/O & MEM DECODE
SD(0..15)
SA(0..19)
SBHE#
IOR#
IOW#
AEN
MEMCS16#
RESET
GND
OWS
+5V
SMEMR#
SMEMW#
16BIT#
LD(0..15)
LA(0..19)
P(0..7)
#CE(0..7)
#WRL(0..3)
#WRH(0..3)
#RD(0..3)
PD(0..6)
T(0..7)
VPP
RESET#
KA(0..15)
SOCKETS
LD(0..15)
LA(0..19)
P(0..7)
#CE(0..7)
#WRL(0..3)
#WRH(0..3)
#RD(0..3)
PD(0..6)
T(0..7)
VPP
RESET#
KA(0..15)
+5V
GND

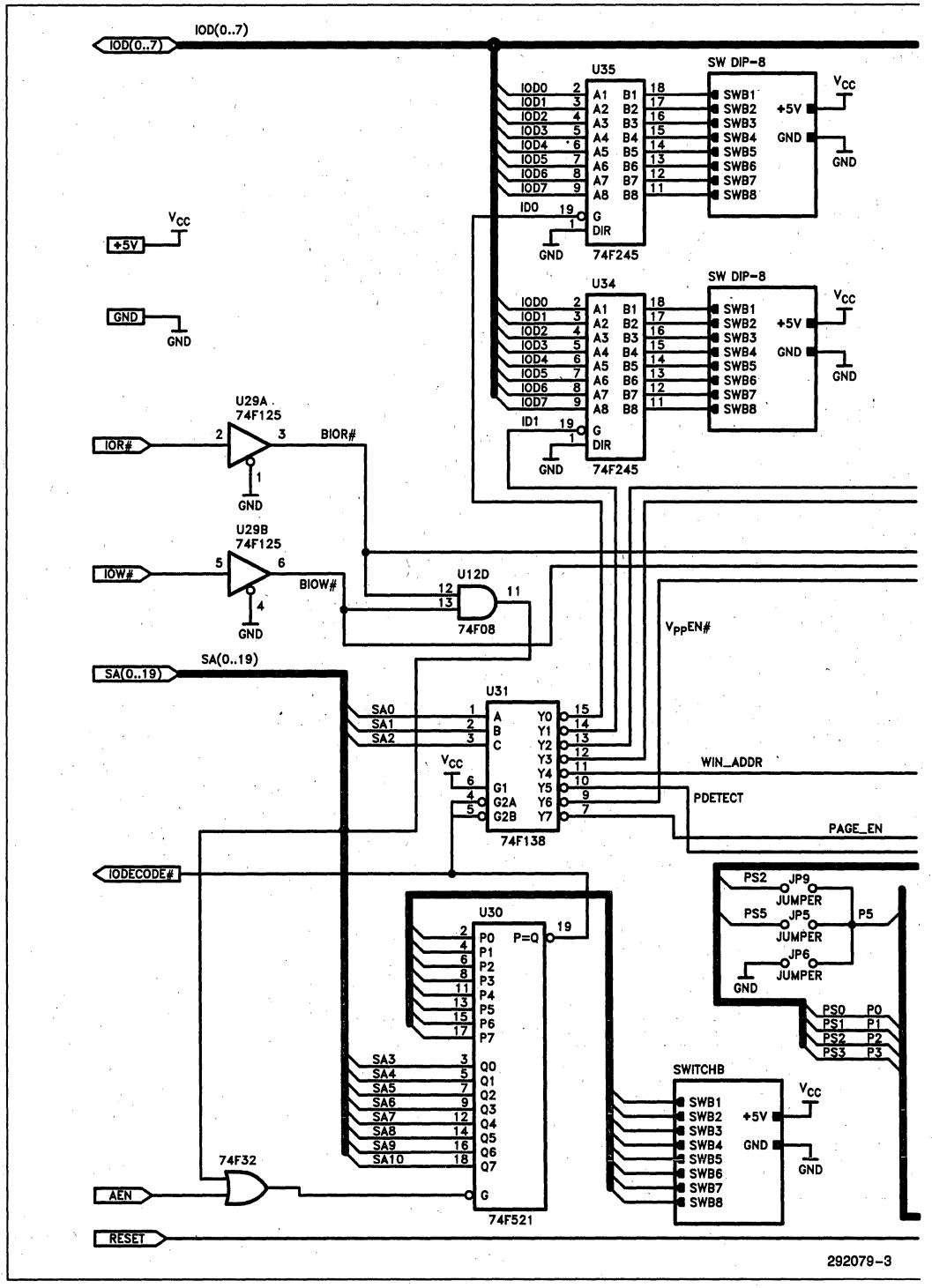
292079-16

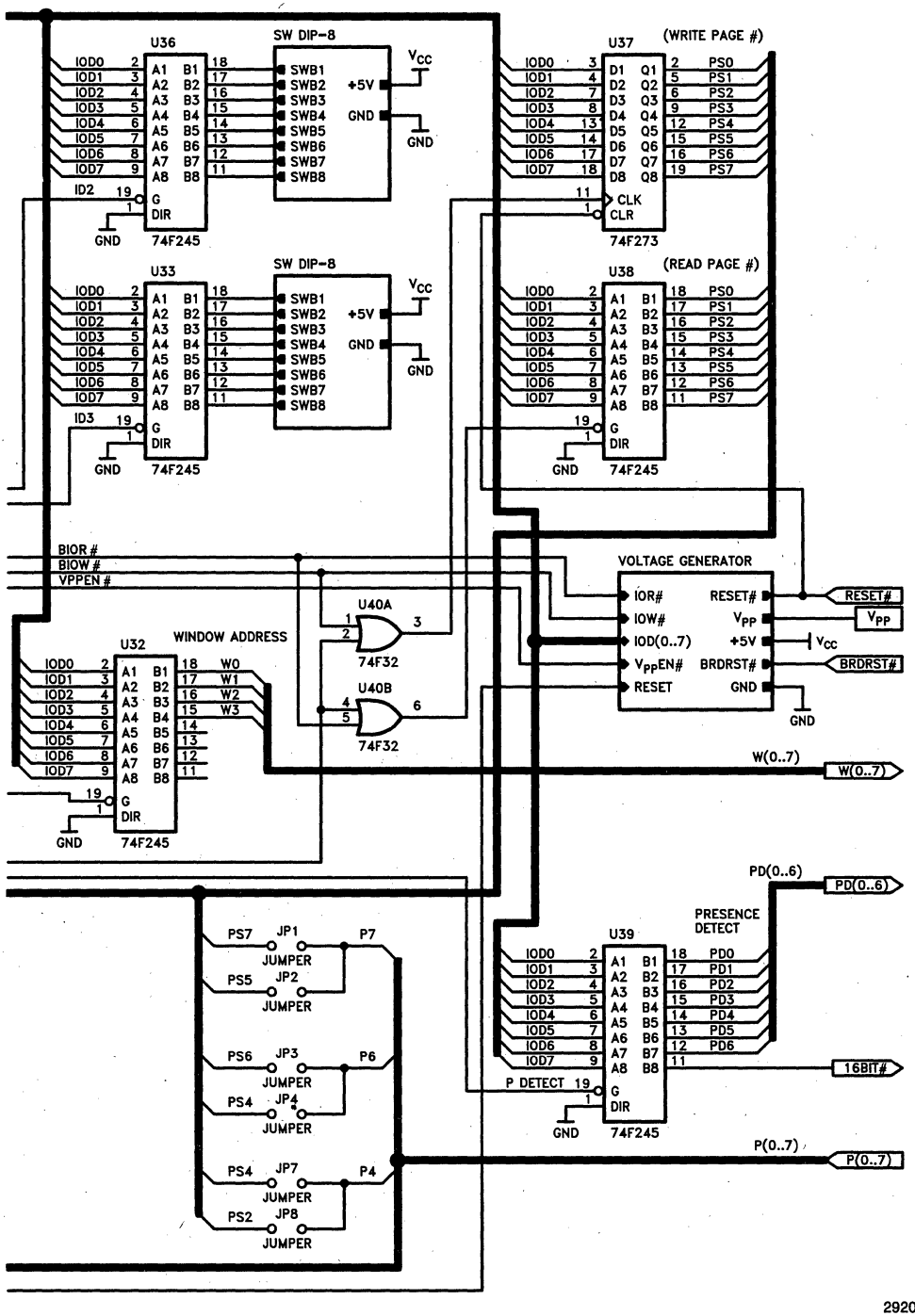

292079-15
SIMM_1_2
SIMM_3_4
VCC
GND
+5V
T(0..7)
RESET#
A(0..14)
D(0..15)
#RD(0..3)
#CE(0..7)
#WEL(0..3)
#WEH(0..3)
VPP
PD(0..6)
P(0..7)
LA(0..19)
LD(0..15)
#WRL(0..3)
#WRH(0..3)
KA(0..15)


SA(0..19)
LA(0..19)
U17
74F244
U18
74F244
U19
74F244
U20
74F244
U16B
74F125
SMEMW#
GND
SBHE#
AEN
RESET
IOR#
IOW#
SMEMR#
U12B
74F08
MEMCS16#
U16A
74F125
U10
74F245
U9
74F245
U11
74F245
LD(0..15)
LOW 8/16 BIT
HIGH 8 BIT
16 BIT
SD(0..15)
Vcc
+5V
R1H
10k
16BIT#
292079-1

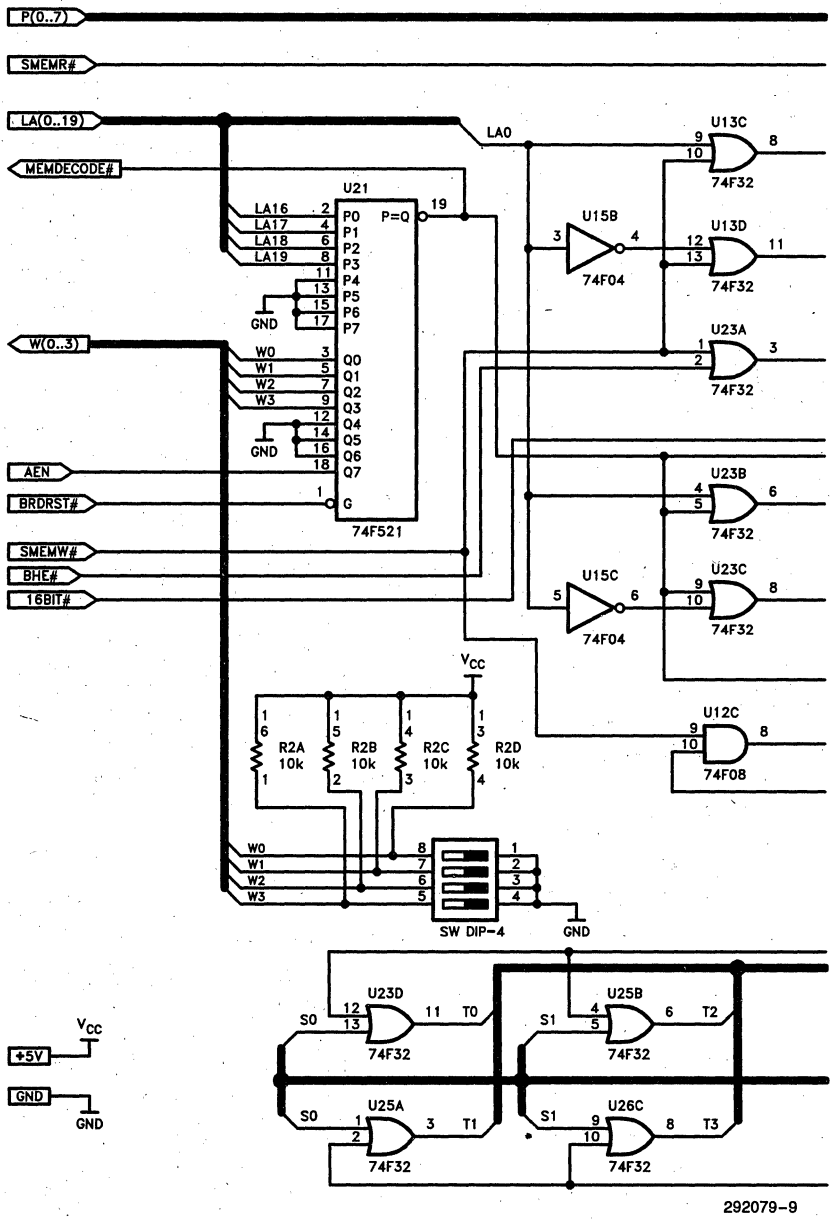

MEM DECODE & SIMM
VCC
+5V
LA(0..19)
AEN
BHE#
P(0..7)
W(0..3)
16BIT#
SMEMW#
SMEMR#
MEMDECODE#
#CE(0..7)
#WRL(0..3)
#WRH(0..3)
T(0..7)
#RD(0..3)
OWS
BRDRST#
GND
#CE(0..7)
#WRL(0..3)
#WRH(0..3)
T(0..7)
#RD(0..3)
OWS
I/O DECODE & VPP
VPP
SA(0..19)
IOR#
IOW#
AEN
RESET
IOD(0..15)
16BIT#
P(0..7)
W(0..3)
BRDRST#
PD(0..6)
RESET#
IODECODE#
SBHE#
PD(0..6)
RESET#
KA(0..15)
LA(0..19)
KA0
U13A
LOW 8/16 BIT
74F32
IODECODE#
U12A
74F08
U15A
74F04
U14A
HIGH 8 BIT
74F10
MEMDECODE#
U13B
16 BIT
74F32
LD(0..15)
16 BIT#

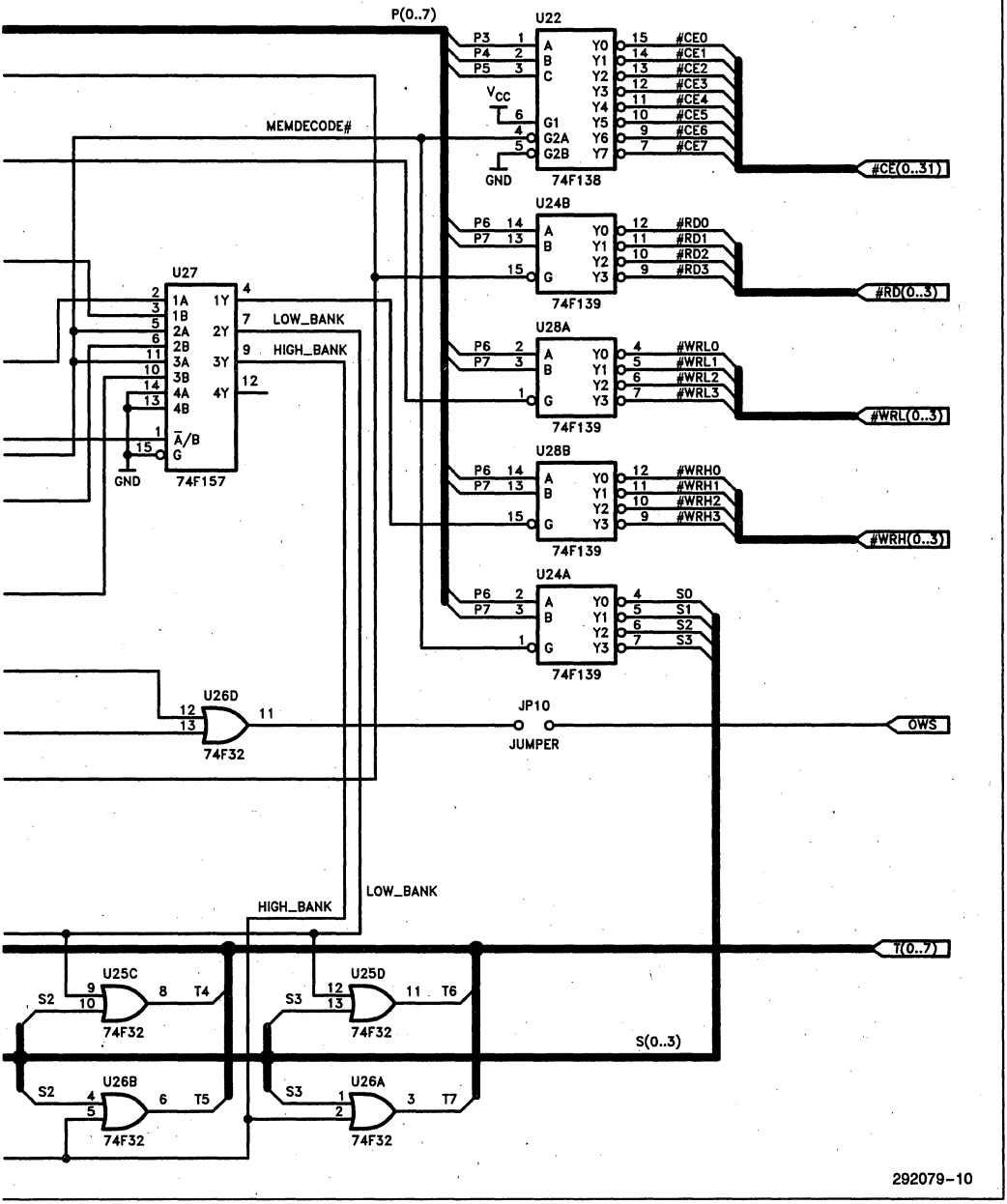
292079-2


IOD(0..7)
U35
SW DIP-8
IOD0
IOD1
IOD2
IOD3
IOD4
IOD5
IOD6
IOD7
IO0
SWB1
SWB2
SWB3
SWB4
SWB5
SWB6
SWB7
SWB8
+5V
GND
74F245
U34
IO1
Vcc
U29A
74F125
IOR#
BIOR#
U29B
IOW#
BIOW#
U12D
74F08
VppEN#
SA(0..19)
U31
SA0
SA1
SA2
74F138
WIN_ADDR
PDETECT
PAGE_EN
IODECODE#
U30
P0
P1
P2
P3
P4
P5
P6
P7
P=Q
SA3
SA4
SA5
SA6
SA7
SA8
SA9
SA10
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
74F521
PS2
JP9
JUMPER
PS5
JP5
P5
JP6
PS0
PS1
PS2
PS3
P0
P1
P2
P3
SWITCHB
74F32
AEN
RESET
292079-3


U36
SW DIP-8
U37
(WRITE PAGE #)
U33
SW DIP-8
U38
(READ PAGE #)
74F245
74F273
IOD0
IOD1
IOD2
IOD3
IOD4
IOD5
IOD6
IOD7
SWB1
SWB2
SWB3
SWB4
SWB5
SWB6
SWB7
SWB8
+5V
GND
VCC
ID2
ID3
G
DIR
CLK
CLR
PS0
PS1
PS2
PS3
PS4
PS5
PS6
PS7
BIOR #
BIOW #
VPPEN #
U40A
U40B
74F32
WINDOW ADDRESS
U32
W0
W1
W2
W3
VOLTAGE GENERATOR
IOR#
IOW#
IOD(0..7)
VPPEN#
RESET
RESET#
VPP
BRDRST#
RESET#
BRDRST#
W(0..7)
PD(0..6)
PRESENCE DETECT
U39
PD0
PD1
PD2
PD3
PD4
PD5
PD6
P DETECT
16BIT#
P(0..7)
JP1
JP2
JP3
JP4
JP7
JP8
JUMPER
P7
P6
P4

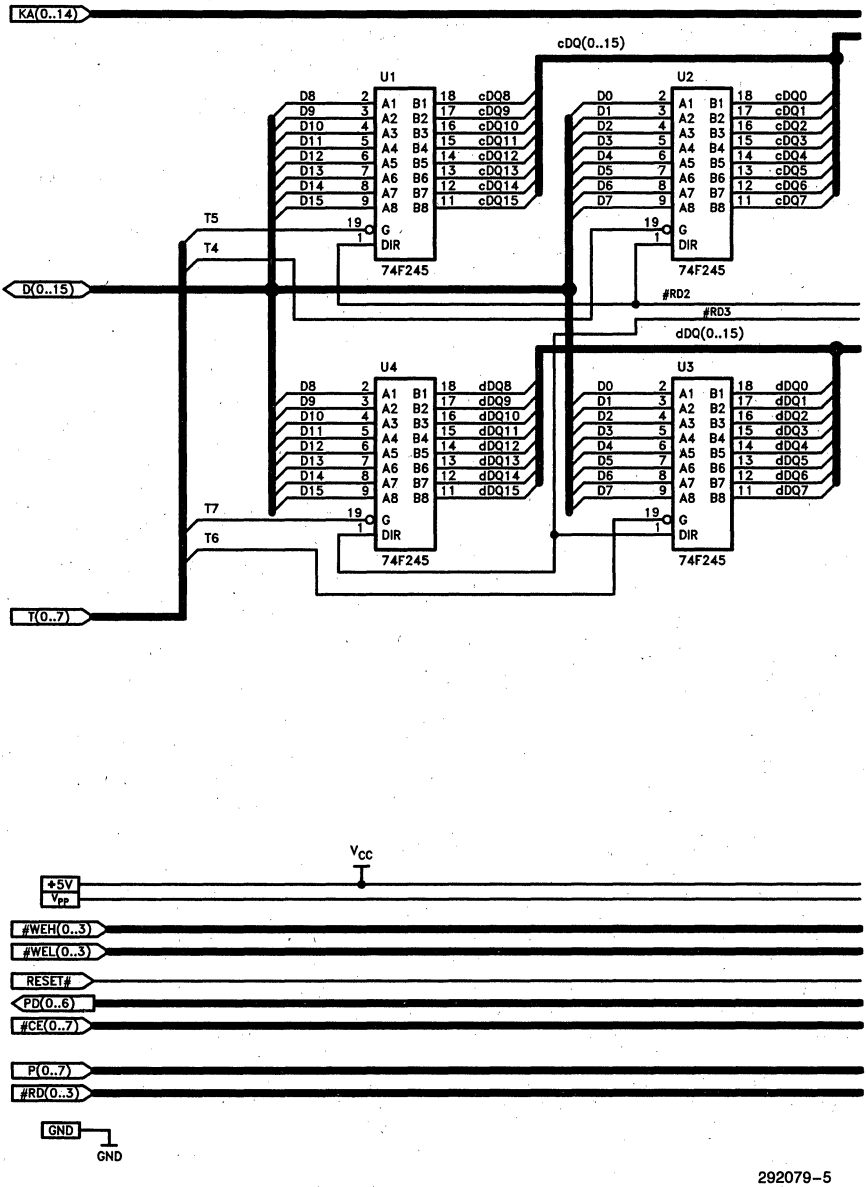
292079-4

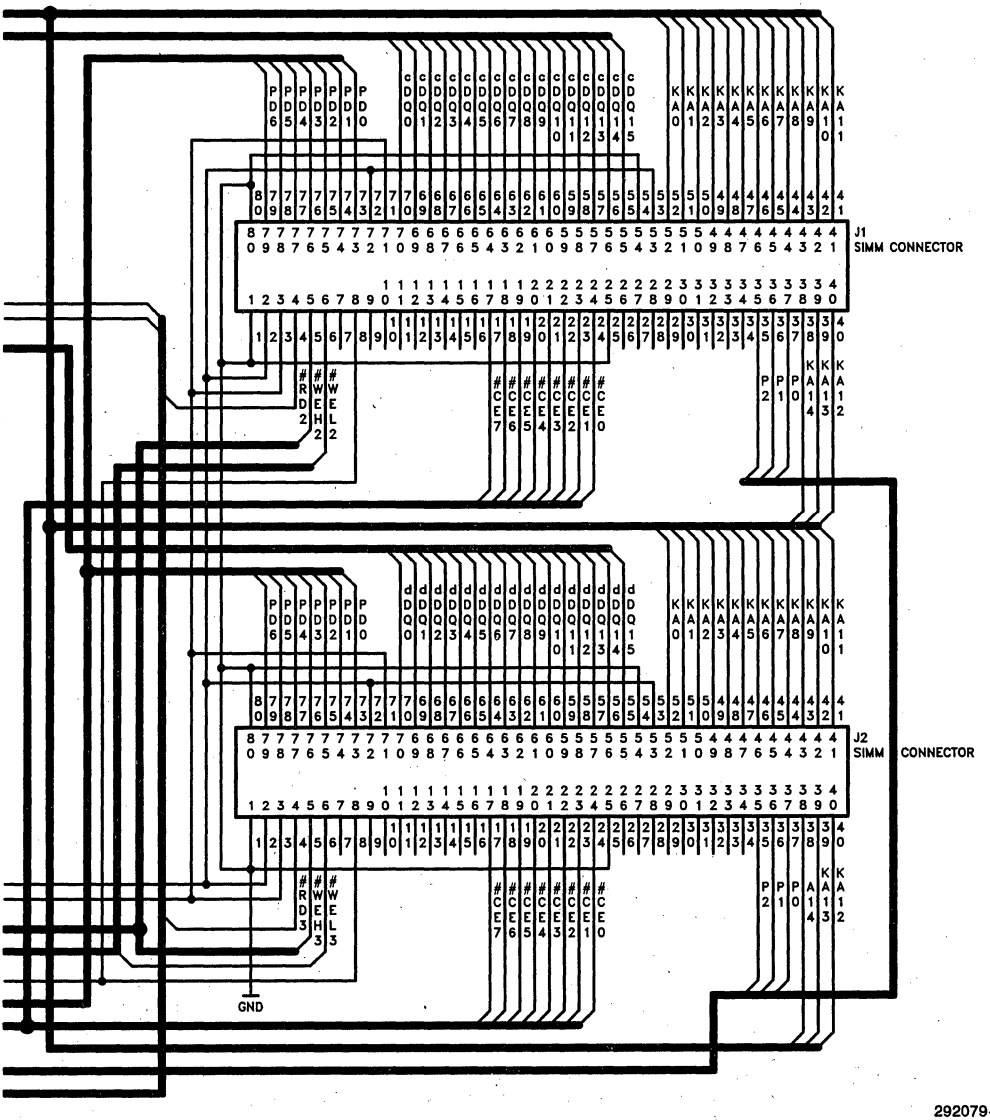

P(0..7)
SMEMR#
LA(0..19)
MEMDECODE#
W(0..3)
AEN
BRDRST#
SMEMW#
BHE#
16BIT#
LA0
U21
LA16
LA17
LA18
LA19
P0
P1
P2
P3
P4
P5
P6
P7
P=Q
19
GND
W0
W1
W2
W3
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
G
74F521
U13C
74F32
U15B
74F04
U13D
74F32
U23A
74F32
U23B
74F32
U15C
74F04
U23C
74F32
U12C
74F08
VCC
R2A 10k
R2B 10k
R2C 10k
R2D 10k
SW DIP-4
GND
U23D
U25B
U25A
U26C
S0
S1
T0
T1
T2
T3
+5V
292079-9

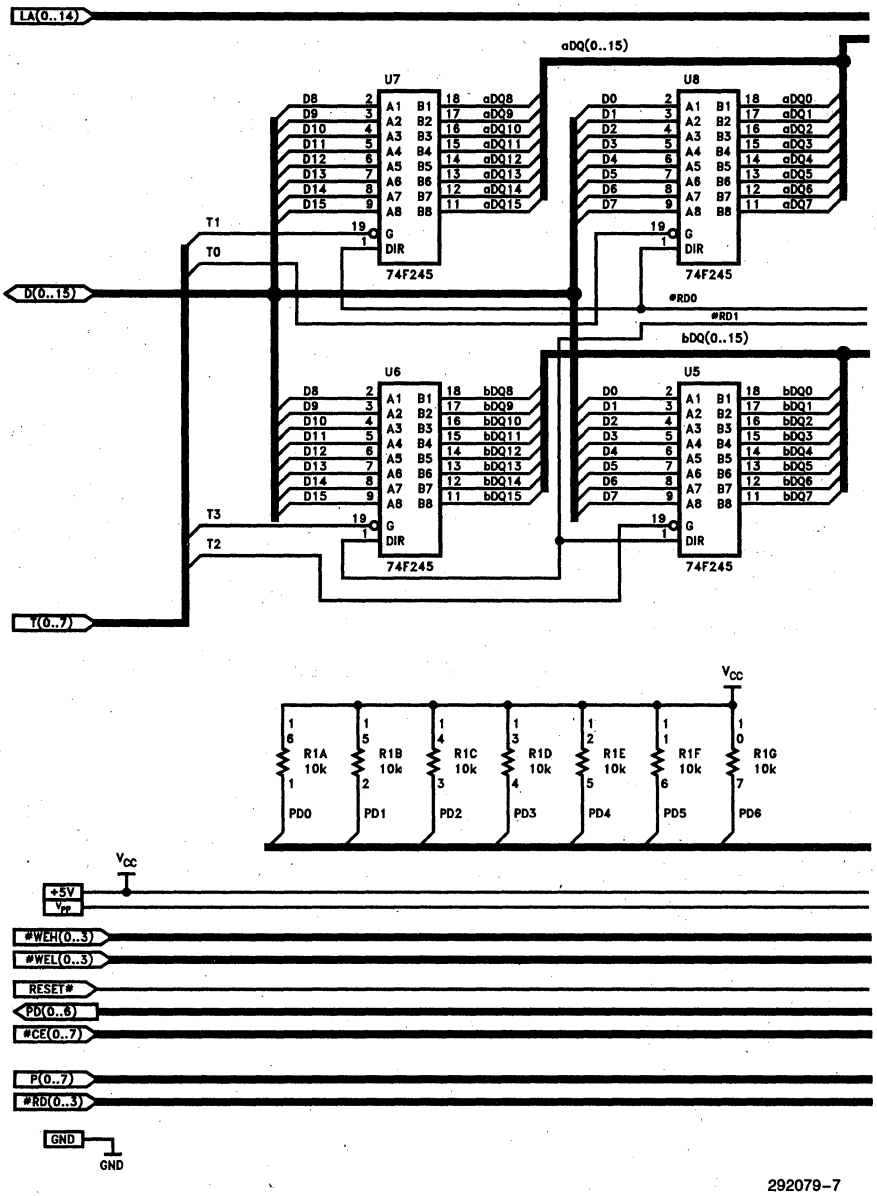

P(0..7)
U22
P3
P4
P5
A
B
C
G1
G2A
G2B
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
VCC
GND
74F138
#CE0
#CE1
#CE2
#CE3
#CE4
#CE5
#CE6
#CE7
#CE(0..31)
MEMDECODE#
U24B
P6
P7
G
74F139
#RD0
#RD1
#RD2
#RD3
#RD(0..3)
U27
1A
1B
2A
2B
3A
3B
4A
4B
A/B
1Y
2Y
3Y
4Y
74F157
LOW_BANK
HIGH_BANK
U28A
#WRL0
#WRL1
#WRL2
#WRL3
#WRL(0..3)
U28B
#WRH0
#WRH1
#WRH2
#WRH3
#WRH(0..3)
U24A
S0
S1
S2
S3
U26D
74F32
JP10
JUMPER
OWS
HIGH_BANK
LOW_BANK
T(0..7)
U25C
U25D
U26B
U26A
T4
T6
T5
T7
S(0..3)

292079-10

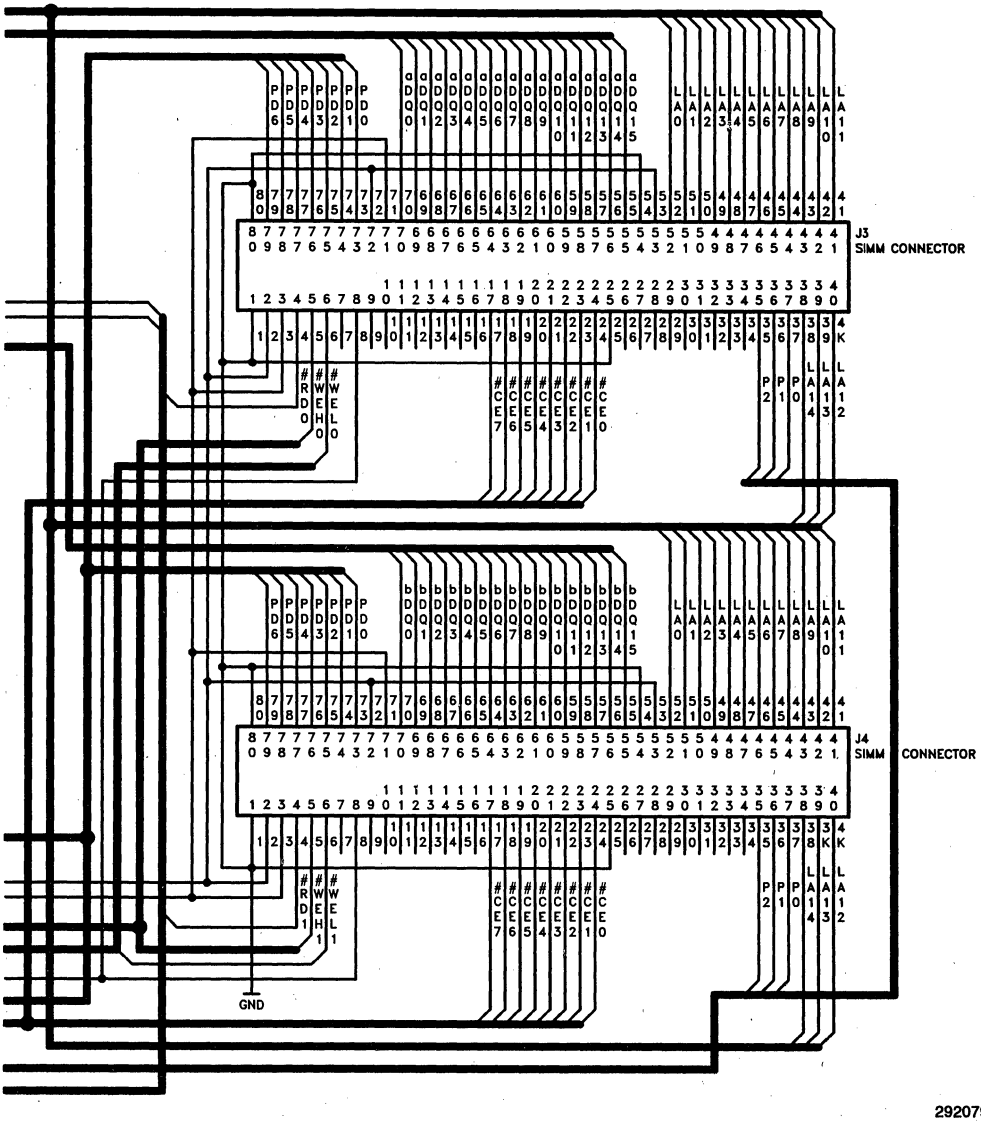
4

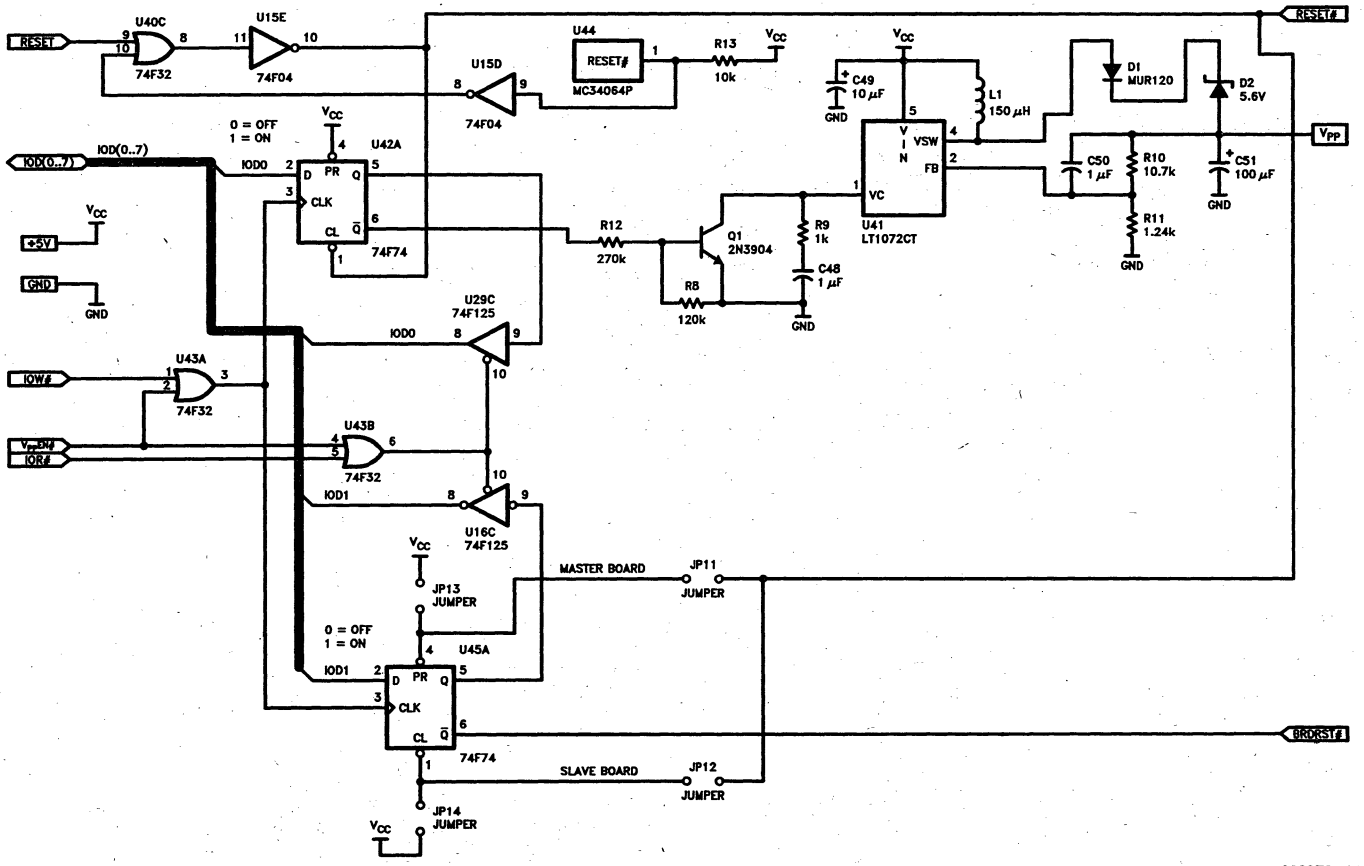

KA(0..14)
cDQ(0..15)
U1
U2
D8 D9 D10 D11 D12 D13 D14 D15
A1 A2 A3 A4 A5 A6 A7 A8
B1 B2 B3 B4 B5 B6 B7 B8
cDQ8 cDQ9 cDQ10 cDQ11 cDQ12 cDQ13 cDQ14 cDQ15
D0 D1 D2 D3 D4 D5 D6 D7
cDQ0 cDQ1 cDQ2 cDQ3 cDQ4 cDQ5 cDQ6 cDQ7
G
DIR
74F245
T5
T4
D(0..15)
#RD2
#RD3
dDQ(0..15)
U4
U3
dDQ8 dDQ9 dDQ10 dDQ11 dDQ12 dDQ13 dDQ14 dDQ15
dDQ0 dDQ1 dDQ2 dDQ3 dDQ4 dDQ5 dDQ6 dDQ7
T7
T6
T(0..7)
Vcc
+5V
Vpp
#WEH(0..3)
#WEL(0..3)
RESET#
PD(0..6)
#CE(0..7)
P(0..7)
#RD(0..3)
GND
GND
292079-5

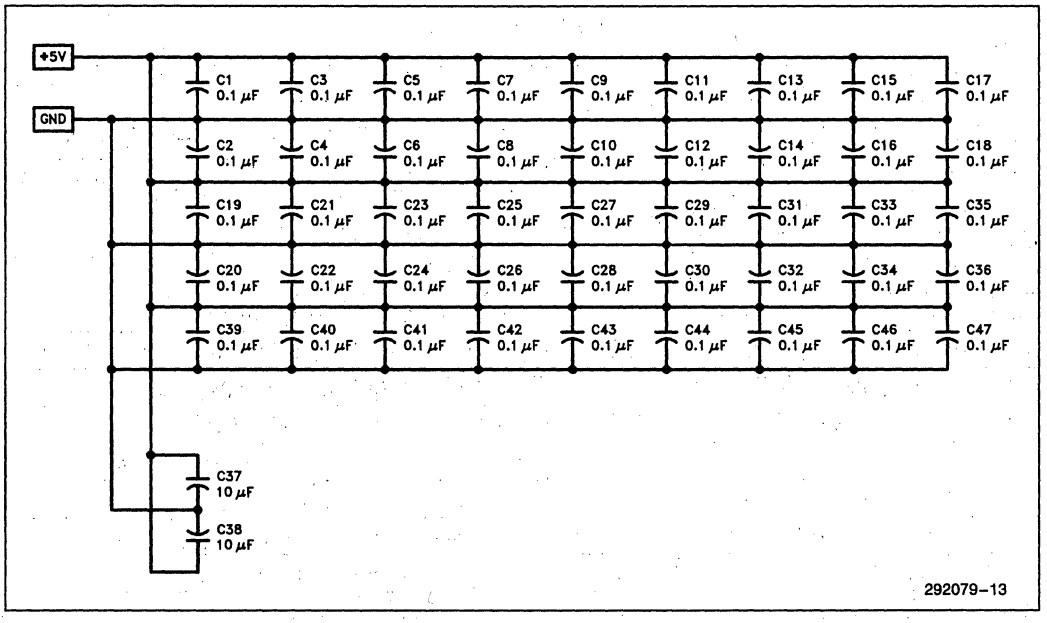

J1
SIMM CONNECTOR
J2
SIMM CONNECTOR
GND
292079-6

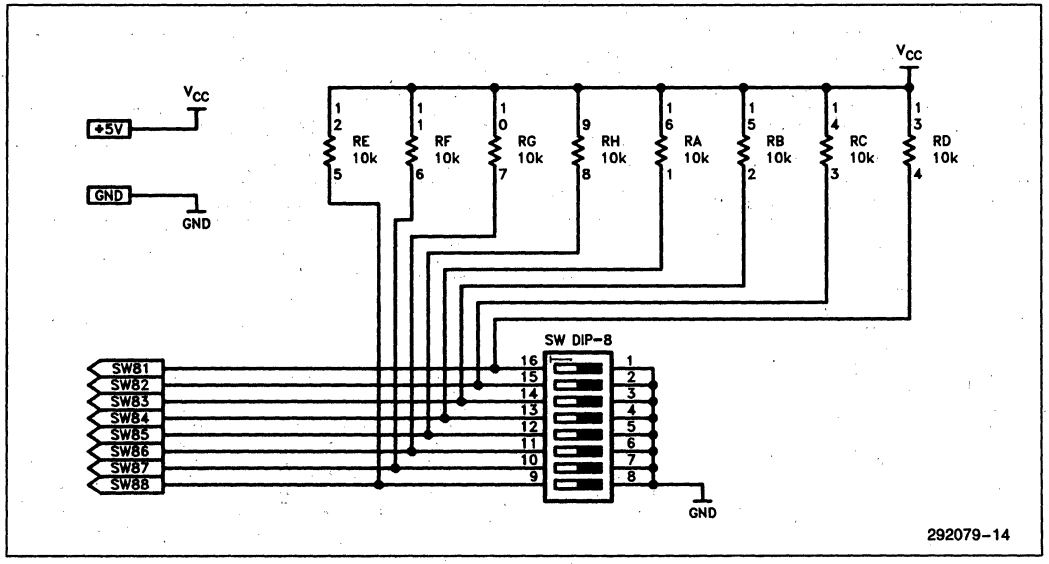

LA(0..14)
aDQ(0..15)
U7
U8
74F245
T1
T0
D(0..15)
#RD0
#RD1
bDQ(0..15)
U6
U5
T3
T2
T(0..7)
VCC
R1A 10k
R1B 10k
R1C 10k
R1D 10k
R1E 10k
R1F 10k
R1G 10k
PD0
PD1
PD2
PD3
PD4
PD5
PD6
+5V
VPP
#WEH(0..3)
#WEL(0..3)
RESET#
PD(0..6)
#CE(0..7)
P(0..7)
#RD(0..3)
GND

292079-7


J3
SIMM CONNECTOR
J4
SIMM CONNECTOR
GND
292079-8


RESET
U40C
74F32
U15E
74F04
U15D
74F04
U44
RESET#
MC34064P
R13
10k
VCC
C49
10 µF
GND
L1
150 µH
D1
MUR120
D2
5.6V
RESET#
VPP
U41
LT1072CT
VIN
VSW
FB
VC
C50
1 µF
R10
10.7k
R11
1.24k
C51
100 µF
0 = OFF
1 = ON
IOD(0..7)
U42A
74F74
IOD0
D
PR
Q
CLK
CL
+5V
GND
R12
270k
Q1
2N3904
R8
120k
R9
1k
C48
1 µF
U29C
74F125
IOD0
U43A
74F32
IOW#
U43B
74F32
IOR#
IOD1
U16C
74F125
JP13
JUMPER
MASTER BOARD
JP11
JUMPER
0 = OFF
1 = ON
U45A
74F74
IOD1
BRDRST#
SLAVE BOARD
JP12
JUMPER
JP14
JUMPER
292079-11


+5V
GND
C1 0.1 μF
C2 0.1 μF
C3 0.1 μF
C4 0.1 μF
C5 0.1 μF
C6 0.1 μF
C7 0.1 μF
C8 0.1 μF
C9 0.1 μF
C10 0.1 μF
C11 0.1 μF
C12 0.1 μF
C13 0.1 μF
C14 0.1 μF
C15 0.1 μF
C16 0.1 μF
C17 0.1 μF
C18 0.1 μF
C19 0.1 μF
C20 0.1 μF
C21 0.1 μF
C22 0.1 μF
C23 0.1 μF
C24 0.1 μF
C25 0.1 μF
C26 0.1 μF
C27 0.1 μF
C28 0.1 μF
C29 0.1 μF
C30 0.1 μF
C31 0.1 μF
C32 0.1 μF
C33 0.1 μF
C34 0.1 μF
C35 0.1 μF
C36 0.1 μF
C39 0.1 μF
C40 0.1 μF
C41 0.1 μF
C42 0.1 μF
C43 0.1 μF
C44 0.1 μF
C45 0.1 μF
C46 0.1 μF
C47 0.1 μF
C37 10 μF
C38 10 μF
292079-13


VCC
+5V
GND
GND
VCC
RE 10k
RF 10k
RG 10k
RH 10k
RA 10k
RB 10k
RC 10k
RD 10k
SW DIP-8
SW81
SW82
SW83
SW84
SW85
SW86
SW87
SW88
GND
292079-14

4

I/O Port Usage for PCAT

Range	Usage
0000–000fh	DMA Controller 1, 8237A
0020–0021h	Interrupt Controller 1, 8259A
0040–005fh	Programmable Timer, 8254
0060–006fh	Keyboard Controller, 8042
0070–007fh	CMOS Real-Time Clock, NMI Mask
0080–009fh	DMA Page Registers, 74LS612
00a0–00a1h	Interrupt Controller 2, 8259A
00c0–00dfh	DMA Controller 2, 8237A
00f0–00ffh	Math Coprocessor
01f0–01f8h	Fixed Disk
0200–020fh	Game Controller
0278–027fh	Parallel Printer Port 2
02b0–027dfh	EGA (Alternate)
02e1h	GPIB (Adapter 0)
02e2–02e3h	Data Acquisition (Adapter 0)
02f8–02ffh	Serial Communications (COM2)
0300–031fh	Prototype Card
0360–036fh	PC Network
0378–037fh	Parallel Printer Port 1
0380–038ch	SDLC Communications

Range	Usage
0390–0393h	Cluster (Adapter 0)
03a0–03a9h	BSC Communications (Primary)
03b0–03bfh	Monochrome/Parallel Printer Adapter
03c0–03cfh	EGA (Primary)
03d0–03dfh	CGA
03f0–03f7h	Floppy Disk Controller
03f8–03ffh	Serial Communications (COM 1)
06e2–06e3h	Data Acquisition (Adapter 1)
0790–0793h	Cluster (Adapter 1)
0ae2–0ae3h	Data Acquisition (Adapter 2)
0b90–0b93h	Cluster (Adapter 2)
0ee2–0ee3h	Data Acquisition (Adapter 3)
1390–1393h	Cluster (Adapter 3)
2390–2393h	Cluster (Adapter 4)
42e1h	GPIB (Adapter 2)
62e1h	GPIB (Adapter 3)
82e1h	GPIB (Adapter 4)
a2e1h	GPIB (Adapter 5)
c2e1h	GPIB (Adapter 6)
e2e1h	GPIB (Adapter 7)

ASSEMBLY LANGUAGE CODE FOR PAGED BOARD

```
;*****************************************************************************
;Locating the Base I/O Address.
;BOARD_NOT_FOUND is an error procedure and is not shown.

    BRD_ID0          dw 4 dup (?)
    Window_Base      dw ?
    Presence_Detect  dw ?
    VPPEN            dw ?
    Page_Number      dw ?

    mov dx,02F8h          ;Set port pointer to 02F8H.

KEEP_LOOKING:
    add dx,8              ;First valid address after adding.
    cmp dx,318            ;Port pointer = 8 less than highest port address?
    jg board_not_found    ;Hey, you forgot to install the board!!!

    in al,dx              ;Read port data into al register.
    cmp al,0Dh            ;Does register = 1st identifier value?
    jne KEEP_LOOKING      ;Not equal --> Not located yet

    inc dx
    in al,dx
    cmp al,0Ah            ;Does register = 2nd identifier value?
    jne KEEP_LOOKING

    inc dx
    in al,dx
    cmp al,01h            ;Does register = 3rd identifier value?
    jne KEEP_LOOKING

    inc dx
    in al,dx
    cmp al,0Eh            ;Does register = last identifier value?
    jne KEEP_LOOKING      ;TOO BAD, you almost had it!

;FOUND-Good Job!
    sub dx,3              ;Restored to base address.
    mov BRD_ID0,dx        ;Save the value in RAM.
```

Locating the Base I/O Address

NOTE:
A review of 8086 asembly language programming fundamentals might be necessary at this point.

4

```
;Locating the Base Memory Address

;*****************************************************************************

;This Information is loaded into a segment register to access data

;*****************************************************************************
    mov ax,0                ;Clear register
    mov dx,Window_Base      ;Port pointer = I/O to read base memory address
    in al,dx                ;Read port
    mov bx,256
    mul ax                  ;Shifts address information left.
    mov es,ax               ;es used as segment register for board.
```

```
;*****************************************************************************
;Determining Memory Capacity

    Density_Lookup_Table dw ?,?,0fffh,7ffh,03ffh
    DENSITY dw ?

    mov ax,0                ;Clears register.
    mov dx,Page_Number      ;Port pointer accesses page number.
    mov al,0
    out dx,al               ;Write a zero to page number hardware.
    mov dx,Presence_Detect  ;Port pointer accesses presence detect pins.
    in al,dx
    and al,23H              ;Clears all but density information.
    cmp al,20H              ;Checks if PD6 is set.
    jng skip_or
    or al,4                 ;If greater than 20H, set bit 2 of al.

;Go to density lookup table, translate value from PD pins, store in RAM
;variable DENSITY. Density value must be 2, 3, or 4 to be valid.
 skip_or:
    cmp al,4
    je okay
    cmp al,3
    je okay
    cmp al,2
    je okay
    jmp Unknown_Device      ;Invalid or no SIMMs present, routine not shown.

;Base address of density lookup table stored in bx register.
    mov bx,offset Density_Lookup_Table
    mov si,ax               ;si register will be pointer into density table
;Density values for 1M-4M, multiples of 1 Kbytes, stored in lookup table.
    mov ax,[bx+si]          ;Density read into ax register
    mov DENSITY,ax
;Read the device identifier. Use the es segment register for the base
;address of the memory.
;28F010 = 0B4h,28F020 = 0BDh
    mov ax,DENSITY          ;Put RAM held density info into ax.
    mov bx,1
    mov es:[bx],90H         ;Write ID command.
    mov bx,es:[bx]          ;Read device identifier.
    cmp bx,0B4h
    je 1MEG
```

Determining Memory Capacity

```
    cmp bx,0BDh
    je 2MEG

    jmp Unknown_device      ;If other than 28F010 or 28F020.

;Divide SIMM density by component density to determine number of components
;on SIMM.

1MEG:
    mov bx,3FFh
    div ax                  ;Divide ax/bx, # of components stored in al.
    jmp NEXT_OPERATION

2MEG:
    mov bx,7FFh
    div ax
    jmp NEXT_OPERATION

;Read from the next SIMM location to verify its presence.
;As an example, assume that the SIMM's density is 1 Mbyte.
;A 1 Mbyte SIMM has 16 pages.
    mov dx,Page_Number
    mov al,16
    out dx,al               ;Switch to Page 16 for next SIMM location.
    mov bx,1
    mov ex:[bx],90H         ;Write ID Command to first device of next SIMM.
    mov ax,es:[bx]          ;Read the identifier. Invalid data = empty socket
                            ;Repeat the process for all SIMMs.

;*****************************************************************************
```

Determining Memory Capacity (Continued)